VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Srovnání webových rámců založených na návrhovém vzoru Model-View-Controller

Comparison of Web Frameworks Based on Design Pattern Model-View-Controller

Student: Bc. Vít Strakoš

Vedoucí diplomové práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2011

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. Vít Strakoš**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 1802T001 Aplikovaná informatika

Téma: Srovnání webových rámců založených na návrhovém vzoru Model-View-Controller
Comparison of Web Frameworks Based on Design Pattern Model-View-Controller

Zásady pro vypracování:

1. Úvod
2. Architektura webových aplikací a Model-View-Controller
3. Analýza stávajícího využití webových rámců ve firmě
4. Návrh případných změn ve využití frameworků ve firmě
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Přílohy

Seznam doporučené odborné literatury:

BROWN, D.; DAVIS, C. M.; STANLICK, S. *Struts 2 in Action.* 1st edition. Greenwich: Manning, 2008. 432 s. ISBN 1-933988-07-X.
CAVANES, C. *Programming Jakarta Struts. 2nd edition.* Sebastopol: O'Reilly Media, 2004. 550 s. ISBN 0-596006-51-9.
WALLS, C.; BREIDENBACH, R. *Spring in Action.* 1st edition. Greenwich: Manning, 2005. 472 s. ISBN 1-932394-35-4.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 26.11.2010
Datum odevzdání: 29.04.2011

Ing. Jan Ministr, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Prohlášení

Místopřísežně prohlašuji, že jsem celou práci,
včetně všech příloh, vypracoval samostatně.

V Ostravě dne 29. dubna 2011 Bc. Vít Strakoš

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu práce Ing. Vítězslavu Novákovi, PhD. za cenné připomínky.

Obsah

1 Úvod

Fenoménem dnešní doby je mezinárodní počítačová síť internet, kterou drtivá většina uživatelů považuje za synonymum internetových stránek, které v současné době přinášejí dříve netušený rozsah interaktivnosti. Dnešním trendem je přesouvat do webové podoby i aplikace, které si ještě před několika málo lety neuměl nikdo představit než jako desktopové. Vždy tomu tak však nebylo, zhruba před pouhými dvěma dekádami byly internetové stránky z dnešního pohledu velmi statické, a to nejen ve smyslu obsahovém (o propojování stránek s databázemi vůbec nemohla být řeč), tak i ve smyslu prezentačním (tehdejším vrcholem pohybu na stránkách by animovaný GIF). Postupně se však tento stav začal měnit, interaktivnost stránek rostla a začaly vznikat složitější webové aplikace, při jejichž tvorbě začalo docházet k opakování určitých programátorských postupů. Programátoři neustále opakovali velmi podobné postupy tvorby kódu, používali totožné struktury aplikace. Postupem času se ukázalo, že nejvhodnější architekturou pro vývoj a údržbu webových aplikací je ta, která je založena na návrhovém vzoru Model-View-Controller. Hlavní myšlenkou této architektury je oddělit aplikační (Model) a prezentační logiku (View), jejichž vzájemnou komunikaci řídí Řadič (Controller). Oddělení vrstev Model a View je absolutní, takže je lze volně substituovat, aniž by to mělo na tu druhou vliv. Tento proces spontánně vyvolal hlad po řešení, které by přineslo zautomatizování těchto rutinních a zároveň vysilujících postupů při tvorbě stránek. A tak zhruba před jednou dekádou, jako odpověď na tuto poptávku, se začaly objevovat první MVC webové rámce (MVC frameworky). Tyto rámce nejen nabízejí výraznou asistenci při aplikaci architektury MVC, ale nabízejí i další služby jako je validace dat či internacionalizace a lokalizace webových aplikací. V dnešní době, kdy se už na scéně zabydlela druhá generace MVC rámců, přerostlo množství a schopnosti těchto rámců mez, kdy již v podstatě není co vyvíjet. Téměř pro každý typ webové aplikace existuje rámec, většinou dokonce open-source s dostatečným počtem využitelných funkcionalit. Dnešní otázkou již tedy není, jak vyvinout rámec pro příslušnou webovou aplikaci, ale který rámec pro ni vybrat.

Cílem této práce je právě popsat strukturu a pokusit se zhodnotit několik nejpopulárnějších webových rámců. A zejména zvážit efektivnost převodu webové aplikace napsané pod starším webovým rámcem Struts 1 na modernější Struts 2 a také popsat samotnou techniku přepisu.

První kapitolou je tento úvod.

Druhá kapitola popisuje strukturu a principy fungování webové aplikace samotné. Podrobněji popisuje význam technologií HTTP, Java Servlet API a samotných MVC rámců. Definuje základní domény působnosti rámců, na jejichž základě jsou postavena hodnotící kritéria jednotlivých rámců. Následuje vlastní hodnocení rámců.

Třetí kapitola zejména analyzuje stav využití rámců v brněnské pobočce firmy CompuGroup Medical Česká republika s.r.o.

Čtvrtá kapitola poskytuje návod pro převod webových aplikací firmy CompuGroup Medical Česká republika s.r.o napsaných pod Struts 1 na Struts 2.

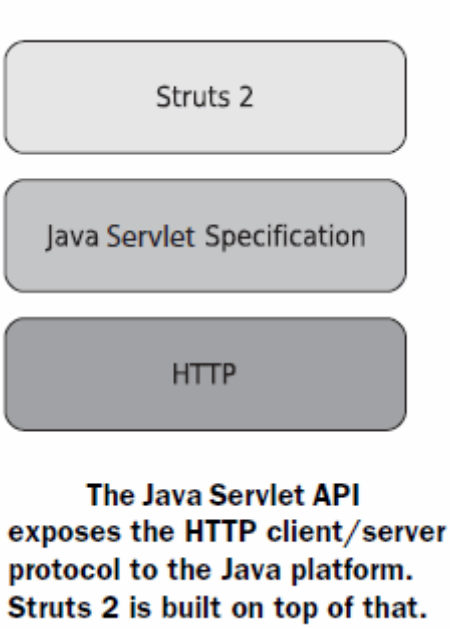
Poslední kapitolou je závěr obsahující shrnutí výsledků práce.

2 Architektura webových aplikací a Model-View-Controller

2.1 Webová aplikace

Webová aplikace je jednoduše aplikace, která běží přes služby WWW. S velkým pokrokem v oblasti rychlosti internetu, připojení a s pokrokem v implementaci client/server technologie, se WWW stává silnou platformou pro budování všech tříd aplikací, od standardních business-orientovaných až po drobné osobní aplikace. Poslední generace webových aplikací musí být plně vybavená a snadno použitelná jako tradiční desktopové aplikace. Ačkoliv roste různorodost webových aplikací, základní „workflow“ těchto aplikací zůstává značně konzistentní, což je ideální příležitost k opětovnému použití. A právě toto opětovné využití („Reused“) je důvodem vzniku webových rámců.

Existují dva hlavní prvky, na kterých jsou postaveny webové aplikace. Jde o řešení jednoduché, kdyby tomu tak nebylo, pravděpodobně by nebylo úspěšné.



Obr. 2.1: Kontext, v němž funguje webový rámec - zde Struts 2 (BROWN & DAVIS, 2008)

Jak je znázorněno na obrázku 2.1, Struts 2 (obecně i jiný webový rámec) běží nad dvěma důležitými technologiemi. V srdci všech Struts 2 aplikace leží klient/server HTTP protokol.

Java Servlet API je pak prostředníkem mezi nízkoúrovňovými HTTP komunikacemi a jazykem Java.

Ačkoliv je možné psát webové aplikace přímým programováním proti Servletu API, není to obecně považováno za správný postup. Struts 2 používá Servlet API, takže tento krok lze vynechat. Nyní následuje stručný popis souvisejících technologií HTTP a Java Servlets.

2.2 Hypertext Transfer Protocol (HTTP)

Většina webových aplikací běží nad HTTP. Tento protokol je bezstavový, umožňuje výměnu zpráv typu klient /server. Za normálních okolností je klientem webový prohlížeč a serverem webový nebo aplikační server. Klient zahájí komunikaci, a to zasláním požadavku na konkrétní zdroj. Zdrojem může být statický HTML dokument, který existuje na serveru v jeho lokálním souborovém systému nebo to může být dynamicky generovaný dokument.
Z pohledu webové aplikace je nejdůležitější vlastností HTTP to, že nebyl původně navržen pro webové aplikace ve smyslu, jaký dnešní vývojáři web aplikací požadují. Byl určen pro „request/response" komunikaci statických HTML dokumentů. Všechny webové aplikace postavené na HTTP musí řešit tento rozpor.

Webovým aplikacím klade HTTP dvě překážky, které je třeba překonat. Je to jeho **bezstavovost** a **textový charakter**. Bezstavové protokoly nesledují vztahy požadavků, které dostávají. Každý požadavek je řešen samostatně, v daném okamžiku je pro server požadavkem jediným. HTTP server nevede žádné záznamy, které by mu umožnily sledovat a logicky propojit více požadavků klienta. Server má sice adresu klienta, ale tu použije pouze pro návrat aktuálně požadovaného dokumentu.

Pokud se vývojář bude snažit budovat složitější webové aplikace s komplikovanějšími případy použití, bude bezstavovost HTTP velmi na obtíž. Vždyť jen obyčejný požadavek na chráněnou komunikaci vede k potřebě autentifikace uživatele. Ten odešle uživatelské jméno a heslo, což musí být určitým způsobem propojeno se všemi ostatními requesty pocházejícími od stejného uživatele během jedné relace.

Stejně tak je nepříjemné, že je HTTP textově založen. Adaptace textově založené technologie na „strongly-typed" technologii, jako je Java, vytváří značnou potřebu napojování dat („data-binding"). V podobě požadavku („request") protokolu HTTP musí být všechna data

reprezentována jako text. A naopak musí být data HTTP requestu namapována na datové typy jazyka Java.

Parametry příchozího požadavku musí být přeneseny do prostředí Java a odchozí odpovědi („response“) musí konvertovat data z Javy zpět do HTTP textové odpovědi. I když nejde o žádný velký programátorský problém, může tvůrcům webových aplikací přinést značnou dávku málo kreativní práce, která je navíc dosti náchylná k omylu.

2.3 Java Servlet API

Java Servlet API pomáhá zmírnit některé obtíže. Tato důležitá technologie funguje jako prostředník mezi HTTP a platformou Java. Základní součásti Servlet API je **servlet**, **request** a **respond objects**.

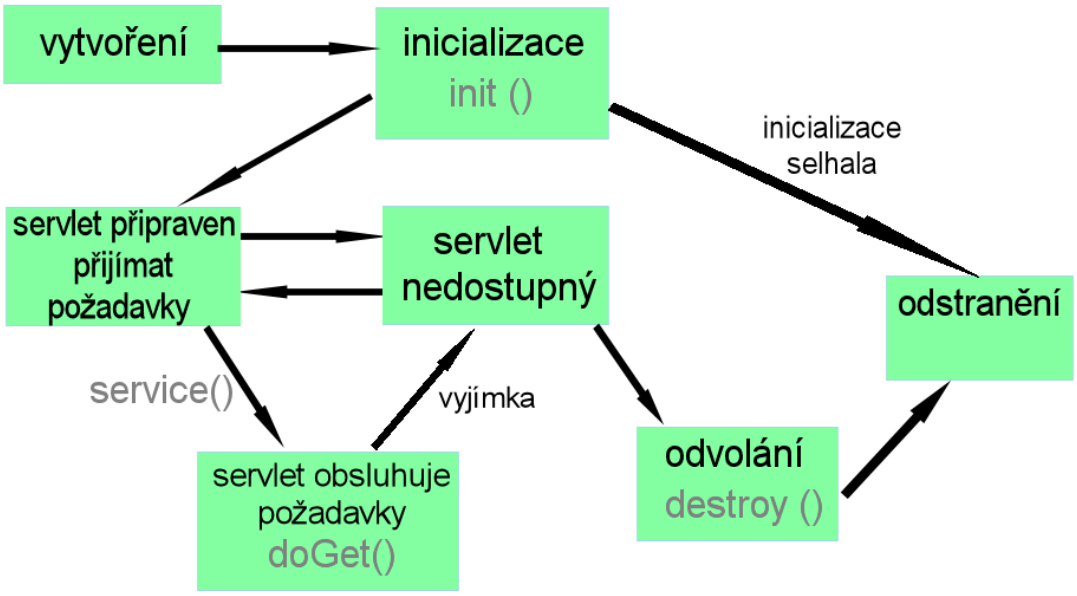
Servlet je Java objekt typu jedináček, jehož základním úkolem je přijmout požadavek a vrátit odpověď následující po algoritmech běžících v pozadí. Objekty requestu zapouzdřují různé jeho detaily včetně všech jeho důležitých parametrů, které jsou zasílány („submit“) prostřednictvím formulářů („form“).

Objekty odpovědi zahrnují součásti jako jsou hlavičky odpovědi a výstupní proud („output stream“), který bude generovat text odpovědi. Jednoduše řečeno: servlet obdrží objekt requestu, prověří správnost jeho dat, udělá v pozadí potřebné operace, pak výsledky zapíše a vrátí odpověď klientovi.

Dříve než lze servlet rozmístit („deploy“) v kontejneru (viz níže), je třeba jej umístit v balíčku („package“) podle určitých standardů. Základní jednotkou balíčkování servletů („servlet-packaging“) je webová aplikace („web application“). Specifikace servletu definuje webovou aplikaci jako „sbírku servletů, HTML stránek, tříd a dalších zdrojů". Zpravidla webové aplikace vyžadují několik servletů k uspokojení požadavků svých klientů. Servlety webových aplikací a jejich zdroje jsou umístěny společně v určité adresářové struktuře a zabaleny v archivu s příponou WAR. Soubor WAR je specializovaná verze souboru Java JAR (BROWN & DAVIS, 2008).

Jakmile je webová aplikace umístěna do balíčku, je třeba ji rozmístit v servlet kontejneru („servlet container“). Nebude se tedy spouštět přímo servlet, ale po rozmístění servletu v kontejneru bude tento kontejner řídit jeho provádění („execution“) voláním různých metod

životního cyklu servletu. Viz obrázky 2.2 a 2.3. Když servlet kontejner obdrží request, musí nejprve rozhodnout, který ze spravovaných servletů by měl požadavek zpracovat.



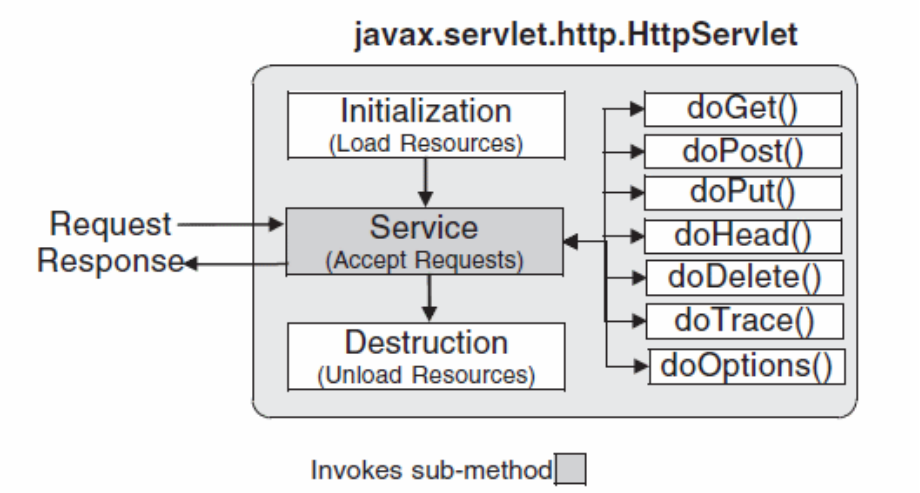
Obr. 2.2: Životní cyklus servletu

Po určení vhodného servletu jsou k dispozici tyto metody životního cyklu servletu:

init() – Inicializace servletu. Objekt servletu je vytvořen a spuštěn v momentě prvního požadavku na servlet. Metoda init() je volána jednou - na začátku životního cyklu. Může například vytvořit spojení s databází či nastavit počáteční hodnoty proměnných.

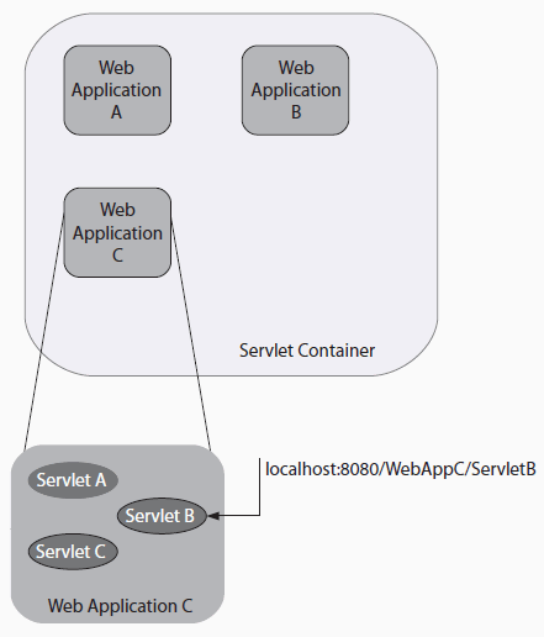
service() – Po proběhnutí metody init() může servlet přijímat požadavky. Požadavek reprezentuje objekt HTTPServletRequest, odpověď objekt HTTPServletResponse. Každý požadavek vytvoří nové vlákno a je volána metoda *service()*. Ta pouze zkontroluje typ požadavku a zavolá příslušnou obslužnou metodu *doGet()*, *doPost()*, *doPut()*, *doDelete()*, *doOptions()* či *doTrace()*. Tato metoda je jádrem činnosti servletu.

destroy() – Tato metoda je volána před tím, než je servlet odstraněn z paměti. K tomu dojde při zastavení serveru. Metoda by měla dělat činnosti jako je uzavření databázového spojení, otevřených souborů, zápis stavu důležitých objektů na harddisk atp.



Obr. 2.3: Životní cyklus servletu (FALKNER, 2004)

Všechny požadavky, bez ohledu na jejich cílové webové aplikace, musí být nejprve řešeny servlet kontejnerem. Servlet kontejner typicky naslouchá na portu 80, kde očekává požadavky. Když na port přijde požadavek, musí servlet být schopen separovat („parse“) jmenný prostor („namespace“) a zjistit, která webová aplikace je vyžadována. Ze jmenného prostoru URL lze stanovit cílenou webovou aplikaci i konkrétní cílený servlet .



Obr. 2.4: Vztah součástí Servlet API - servletů, webových aplikací a servlet kontejneru (BROWN & DAVIS, 2008)

Obrázek 2.4 ukazuje situaci nasazení tří webových aplikací do jednoho kontejneru.

Další neméně významnou funkcí, kterou Servlet API poskytuje, je mechanismus umožňující sdružovat skupiny požadavků jednoho klienta. Jak bylo vysvětleno výše, HTTP je bezstavový, tudíž tuto funkcionalitu nezvládá. Bez této schopnosti servletu bychom byli nuceni neustále z cookies separovat textové řetězce vymezující danou konkrétní session.

Na rozdíl od session mechanismu, Servlet API neposkytuje higher-level funkcionalitu. Přímo totiž zapouzdřuje jednotlivé požadavky a odpovědi HTTP komunikace do sady objektů. To znamená, že programátor nemusí analyzovat příchozí HTTP požadavky sám, místo toho obdrží úhledný objekt již zabalený v Javě. Na základě této skutečnosti mluvíme o Servlet API jako o „**infrastructured-level technologii“** v nabídce dnešních webových aplikací. Tato technologie poskytuje solidní základnu pro tvorbu robustních aplikací. Technologie Servlet API sice zajišťuje vše potřebné pro běh moderních webových aplikací, ale neřeší některé záležitosti, jejichž absence řešení zapříčinily vlastní vznik webových rámců.

2.4 Pole působnosti webových rámců

Technologie Servlet API řeší tzv. low-level klient/server komunikaci. Existuje však mnoho úkolů, které všechny webové aplikace musí řešit a které jdou nad rámec možností Servlet API. Jedná se zejména o rutinní požadavky na vyšší (aplikační) úrovni, jež tvoří hlavní pole působnosti webových rámců. Zároveň tyto aspekty poslouží v dalším průběhu práce jako podklady pro vytvoření kritérií sloužících k rozlišení funkčnosti a vhodnosti konkrétních rámců.

Jde zejména o:

- Napojení parametrů požadavku („binding request parameters“) na Java datové typy.
- Ověření („validating“) dat.
- Volání business logiky.
- Volání datové vrstvy.
- Ztvárnění prezentační vrstvy.
- Internacionalizace a lokalizace.
- Testovatelnost aplikací.

2.4.1 Napojení parametru požadavku a ověření dat

Jak již bylo výše zmíněno, je HTTP textově založený protokol, a proto musí být parametry jeho požadavku v textové podobě. Když tyto parametry vstupují do webové aplikace, musí být převedeny na odpovídající nativní Java datový typ. Parametry požadavku jsou stále ve formě řetězce („String“). Konvertování těchto Stringů do Java datových typů je sice celkem snadné, ale může být časově náročně a náchylné k chybám. Převod na primitivní datové typy je zdlouhavý, převod do složitějších typů je složitý i zdlouhavý.

A samozřejmě je nutné ověření dat pře jejich vstupem do systému.

Existují dvě úrovně validace:

1) Řetězec musí odpovídat potřebnému Java datovému typu (PSČ neobsahuje nečísla).
2) Poté musí být data validována na vyšší logické úrovni („higher-level logic“).
   - V případě PSČ jeho hodnota spadá do definovaných možných hodnot.
   - E-mailová adresa odpovídá povolené syntaxi atp.

Ruční psaní kódu ošetřující tyto aspekty v každé aplikaci vede k otrocké práci programátora.

2.4.2 Volání business logiky a datové vrstvy

Po napojení parametrů požadavků a ověření dat je možné se zaměřit na samotný běh aplikace, kdy většina požadavků volá business logiku a datovou vrstvu. Specifika těchto volání se liší od aplikace k aplikaci, nicméně dvě zobecnění lze vyvodit:

- Vzdor rozdílnostem v detailech těchto jednotlivých volání, lze na základě jejich zobecnění vytvořit jednotný vzor workflow („pattern workflow“). Zpracování každého požadavku se skládá ze sledu prací, které je třeba vykonat. Toto je jádrem činnosti akčně orientovaných (action-oriented) frameworků.

- Logika a funkcionalita volání business logiky a datové vrstvy představuje krok mimo webově orientovanou aplikaci. Jde totiž o jediné činnosti, které webová aplikace musí udělat při zpracování svých požadavků a zároveň vůbec nesouvisí s existencí aplikace jakožto webové (spíše se jedná o funkcionalitu desktopových aplikací). Je-li aplikace dobře navržena, obchodní logika a datová vrstva zcela zakrývají skutečnost, že byly spuštěny z webové aplikace či desktopové aplikace. Pozoruhodné tedy je, že ačkoliv všechny webové aplikace musí tato volání vykonat, stojí tato volání mimo workflow webové aplikace (BROWN & DAVIS, 2008).

2.4.3 Ztvárnění prezentační vrstvy a internacionalizace

Dalo by se říci, že prezentační vrstvou webové aplikace je pouze dokument HTML. Nicméně s rostoucím množstvím složitých JavaScriptů, plnokrevných CSS a jiných vestavěných technologií je třeba toto tvrzení poopravit. Ruku v ruce s rostoucí složitostí uživatelského rozhraní jde požadavek na internacionalizaci aplikací. Internacionalizace umožňuje vybudovat jednoduchou („single“) web aplikaci, která nabízí každému uživateli webové aplikace jeho vlastní lokalizované prostředí (jazyk, formátování data, času a měny).

Všechny tyto výše nastíněné problémy jsou vhodnými kandidáty znovupoužití. A tudíž jsou vhodnými adepty na zahrnutí do řešení webových rámců. Tyto vznikly právě proto, aby nahradily úmornou a stále se opakující práci programátorů webových aplikací.

2.4.4 Testovatelnost aplikací

Otázka schopnosti testovat rychle a pohodlně aplikaci kdykoliv během vývoje je samozřejmě velmi důležitý aspekt mající vliv nejen na celkový čas vývoje, ale zejména na jeho náklady. Schopnost rychle a pohodlně testovat souvisí s mírou schopnosti webového rámce spustit aplikaci mimo kontejner.

2.5 Rámec, návrhový vzor Model – View – Controller (MVC)

2.5.1 Rámec

Chce-li programátor vytvořit výkonné webové aplikace, potřebuje veškerou pomoc, která je dostupná. Pokud nechce trávit hodiny při řešení úkolů popsaných v předchozích částech této práce, musí použít některý z desítek webových rámců.

Ve své nejjednodušší formě je rámec souborem tříd a rozhraní, které v programu společně řeší určitý typ problému (CAVANES, 2004, str. 11).

Co je tedy funkcí webového rámce?

- Rámec se snaží o zevšeobecňování společných úkolů a pracovních postupů specifické oblasti.
- Rámec se snaží poskytovat platformu, na které aplikace poběží rychleji, což zajišťuje automatizací zdlouhavých úkolů a také vytváří elegantní architektonické řešení běžného workflow webové aplikace.

2.5.2 Návrhový vzor Model – View – Controller (MVC)

Všechny webové rámce spojuje skutečnost, že jsou postaveny na návrhovém vzoru MVC. Kořeny lze najít již v době uvedení technologie Java Server Pages. JSP jsou v podstatě HTML stránky, do kterých jsou vložené speciální tagy obsahující Java zdrojový kód. Pro vkládání Java kódu je využíváno speciálních skriptovacích značek:

- Výrazy ve tvaru *<%= výraz %>*.
- Skriptlety ve tvaru *<% kód %>*.
- Deklarace ve tvaru *<%! kód %>*.
- JSP komentáře ve tvaru *<%-- JSP komentář --%>*.

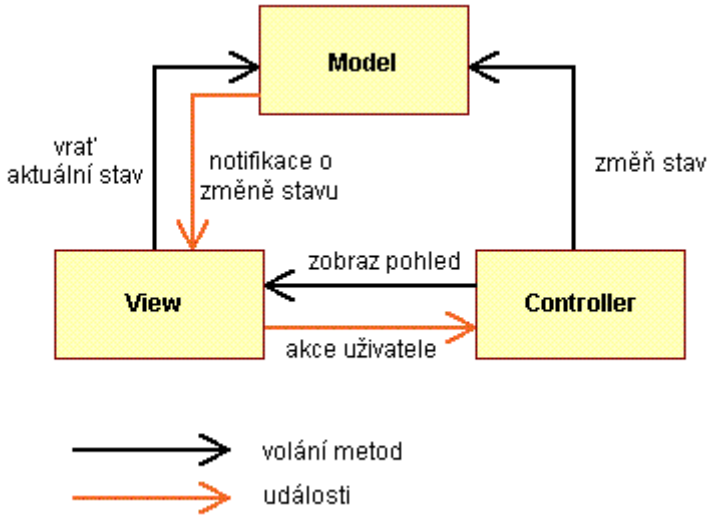
Tento způsob zápisu kódu vedl k tomu, že programátoři psali všechen kód aplikace přímo do JSP, což mělo za následek nerozlišování prezentační a aplikační logiky. Kód volající určitou business logiku byl promíchán s kódem, který měl za úkol výsledky této business logiky zobrazit. Výsledkem této praxe byla nulová znuvupoužitelnost kódu.

Reakcí bylo JSP 1.1, které zavedlo sdružování tagů do knihoven („library“). Tag se skládá ze dvou částí: vlastní zápis podobný HTML elementům a odpovídající Java třída. Tímto oddělením je zaručena znuvupoužitelnost Java kódu.

Ruku v ruce s JSP 1.1 se na přelomu tisíciletí objevily první frameworky založené na návrhovém vzoru MVC. Jedním z prvních a dodnes jedním z nejrozšířenějších byl Struts.

Hlavním smyslem návrhového vzoru MVC je oddělit logiku aplikační (business logika – vrstva Model) od prezentační (uživatelské rozhraní, interakce s uživatelem - vrstva View). Z čehož vyplývá, že není omezen pouze pro využití při tvorbě webových aplikací, ale je vhodný i pro jiné, zejména interaktivní aplikace.

Důležitou vlastností je naprostá nezávislost vrstvy View na vrstvě Model, což přináší možnost snadné zaměnitelnosti vrstvy View. Viz obrázek 2.5.



Obr. 2.5: Schéma fungování MVC – tok událostí a volání metod (Sevrjukov, 2007)

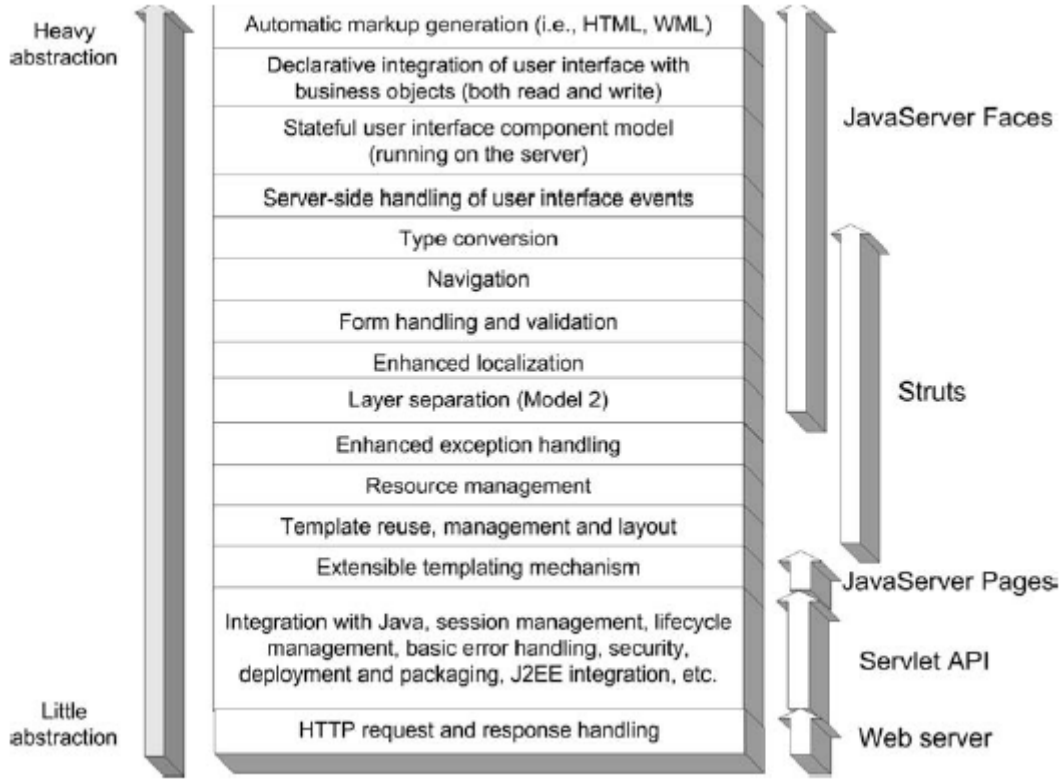
Vrstva Model: Zajišťuje business logiku aplikace, jde tedy o jádro činností aplikace. Díky ní lze rovněž přistupovat k datům. Často jde o nejsložitější vrstvu aplikace rozpadající se na logické subkomponenty.

Vrstva View (Pohled): Představuje prezentační logiku aplikace. Zajišťuje interaktivitu s uživatelem (mimojiné zajišťuje získávání dat od uživatele) a snaží se ve formátu vhodném pro uživatele odprezentovat výsledky procesů business logiky. Existuje i eventualita existence více vrstev View, což má význam při potřebě prezentovat stejná data různým uživatelům či na různých zařízeních.

Vrstva Controller (Řadič): Řídí komunikaci s uživatelem a také mezi komponentami vrstev Model a View. Reaguje na události přicházející z vrstvy View (akce uživatele), na jejich základě upravuje stav vrstvy Model a vybírá pohled vhodný pro uživatele.

2.6 Členění webových rámců MVC do základních kategorií

Za základní hledisko členění MVC webových rámců se dá považovat míra jeho abstrakce nad HTTP protokolem (samozřejmě se frameworky liší i svou vnitřní architekturou). Čím větší je úroveň abstrakce daného rámce nad HTTP protokolem, tím méně se musí jednotlivé vrstvy o něj zajímat. Viz obrázek 2.6.



Obr. 2.6: Míra abstrakce některých rámců nad HTTP protokolem (MANN, 2004)

Historicky starší a z pohledu budoucnosti méně perspektivní jsou rámce s menší mírou abstrakce nad protokolem HTTP. Pro tyto rámce se vžilo označení **požadavkem** (protokolu HTTP) **řízené** („Request based“). Rámce s vysokou mírou abstrakce nad protokolem HTTP označujeme jako **komponentně orientované** („Component based“). V následující tabulce 2.1 jsou shrnuty základní vlastnosti, výhody a nevýhody obou skupin.

Webové rámce	Požadavkem řízené	Komponentně orientované
příklady	Struts1, Struts 2, Spring MVC, Stripes	Java Server Faces, Tapestry
míra abstrakce nad HTTP	malá	velká
zaměření na aplikace	stránky s přesně daným vzhledem	vizuální komponenty (pohyb myši, stisk klávesy) aplikace s GUI
práce s	URL, parametry requestu HTTP	komponenty, události („Events")
budoucnost	méně perspektivní	perspektivní
výhody	velká rozšířenost, znalost rámců (hlavně Struts) → přechod ze Struts do Struts 2 jednodušší	efektivní práce na úrovni komponent, událostí
nevýhody	práce na úrovni URL, HTTP méně efektivní	náročnost zvládnutí technologie

Tab. 2.1 Srovnání webových rámců požadavkem řízených a komponentně orientovaných

Další okolností, jež může hrát velkou roli při výběru vhodného webového rámce je otázka jeho míra jeho dostupnosti (proprietárnost či open-source). Zde není odpověď zcela jednoznačná. Proprietárnost např. rámce Java Server Faces s sebou přináší zároveň výhodu jisté standardizovanosti, což přináší zejména výhodu v podobě velké podpory nástrojů. Naproti tomu open-source řešení je oproti proprietárním řešením ze své podstaty nezávislé na úrovni implementace standardu a v případě řešení nějaké nestandardní situace při tvorbě aplikace obecně nabízí lepší řešení než řešení proprietární.

2.7 Shrnutí základních výhod a nevýhod použití frameworků MVC

2.7.1 Výhody

Výhody frameworku MVC jsou tyto:

- Ušetření času, nákladů: již v okamžiku volby práce s rámcem MVC spoříme čas i náklady tím, že samotný framework je již částečné funkční aplikace, kterou využijeme ve vlastní aplikaci.
- Vyšší spolehlivost aplikace: V případě implementace určité funkcionality vlastním programátorským řešením je určitě riziko nesprávné implementace větší než v případě použití již ověřeného řešení rámce MVC.
- Výhody plynoucí z oddělenosti vrstev:
    - Snadnější paralelní vývoj jednotlivých vrstev odděleně a z toho plynoucí úspora času.
    - Formální přehlednost jednotlivých vrstev umožňující jejich snadnější modifikovatelnost (zejména u prezentační vrstvy).
    - Jednoduchá náhrada každé vrstvy bez vlivu na vrstvy ostatní.
    - Vrstva Pohled existující ve více možnostech podle požadavku různých uživatelů.
    - Možnost znuvupoužitelnosti vrstev v budoucích aplikacích.
    - Jednodušší servis aplikace a korekce chyb.

2.7.2 Nevýhody

Nevýhody frameworku MVC jsou tyto:

- Studijní nároky: Každá technologie (tedy i rámce MVC) klade určité nároky na programátory z hlediska jejího zvládnutí a schopnosti použít ji.
- Vznik závislosti aplikace na frameworku: Je-li framework jednou do aplikace implementován, je nutné při dalším vývoji či údržbě aplikace respektovat principy jeho fungování, protože je nedílnou součástí aplikace.
- Zhoršení výkonu aplikace
    - Nutnost implementace knihoven: Každý framework využívá specifické knihovny, které je nutné včlenit do aplikace.
    - Složitější struktura objektů (rámce vytvářejí různé pomocné objekty).

2.8 Východiska hodnocení rámců MVC

Kritéria hodnocení jednotlivých vybraných rámců budou založeny především na kapitole 2.4 „Pole působnosti webových rámců“. V této kapitole byly uváděny tyto aspekty, které byly označeny za hlavní původce samotné potřeby vzniku webových rámců:

- Napojení parametrů požadavku na Java datové typy.
- Ověření dat.
- Volání business logiky.
- Volání datové vrstvy.
- Ztvárnění prezentační vrstvy.
- Internacionalizace a lokalizace.
- Testovatelnost aplikací.

Dále každý rámec zařadíme dle kapitoly 2.6 „Členění webových rámců MVC do základních kategorií“ buď do skupiny požadavkem řízeného nebo komponentně orientovaného rámce.
Nakonec budou hodnocena dvě uživatelská kritéria, jejichž splnění není sice pro úspěšný běh webové aplikace nezbytně nutné, ale ledacos napoví o schopnosti psát aplikace „user-friendly“, kdy zejména v případě kritéria hodnotícího chování aplikace po stisknutí tlačítek *vzad*, *vpřed* a *reload* v prohlížeči může dojít až ke zhroucení aplikace.

2.8.1 Kritéria hodnocení

Kritéria hodnocení jednotlivých vybraných rámců jsou následující:

1) **Druh rámce**: Požadavkem řízený nebo komponentně orientovaný.

2) **Ověření dat**: Zhodnocení náročnosti ověření dat zadaných uživatelem. Ideálně se validuje jak na straně uživatele vhodným skriptovacím jazykem, tak na straně serveru. Tento postup povede nepochybně ke snížení nároků na běh aplikace.

3) **Ztvárnění prezentační vrstvy**: Jaké technologie prezentační vrstvy jsou k dispozici.

4) **Internacionalizace-lokalizace**: Schopnost nabídnout jednotnou aplikaci uživatelům různé národnosti, která bude zároveň schopna tyto uživatele identifikovat a nabídnout každému z nich jeho specifické národní prostředí (měna, formátování data, atp.).

5) **Míra samostatné testovatelnosti aplikace**: Otázka schopnosti testovat aplikaci mimo kontejner, ve kterém běží.

6) **Bookmarks-friendly**: Lze uložit do samostatné záložky každou stránku průběhu webové aplikace? V podstatě jde o odpověď na otázku, zda příslušný webový rámec používá nad HTTP request metodu post či get.

7) **Reakce na stisk specifických tlačítek prohlížeče**: Zvládne aplikace korektně pokračovat i po stisku tlačítek *vpřed*, *vzad* či *reload* ve webovém prohlížeči?

2.9. Hodnocení vybraných rámců

2.9.1 Apache Struts

Domovský web: http://struts.apache.org/

Apache Struts je open-source webový aplikační framework pro vývoj Java EE webových aplikací. Původně byl vytvořen Craigem Mcclanahanem, který jej daroval nadaci Apache v květnu 2000. Dříve byl umístěn pod projekty Apache Jakarta a známý jako Jakarta Struts, v roce 2005 se stal jedním ze stěžejních projektů Apache. Není zároveň tak robustní jako modernější komponentně orientované rámce, čímž zároveň vyžaduje relativně méně času na pochopení a použití v praxi.

Je-li řeč o rámci Struts, je třeba nejprve zmínit existenci dvou odlišných verzí, které však dodnes existují vedle sebe. Původní (Jakarta) Struts (dnes označována jako verze 1 či classic) byla vydána v červnu 2001. Jelikož to bylo v samém počátku období, kdy vznikala poptávka po frameworcích webových aplikací, došlo ihned k jeho značnému rozšíření a dodnes si zachovává přes své stáří a nedokonalost vedoucí postavení mezi všemi MVC frameworky. Stále v podstatě funguje jako standard MVC rámců, se kterým je každý nový framework srovnáván a jehož slabiny se snaží opravit.

Struts 2 byl oficiálně uvolněn na sklonku roku 2006. Jedná se o syntézu Struts a jiného rámce Webwork, který byl vyvíjen od roku 2002 s cílem vyřešit hlavní slabé stránky původního Struts a zároveň začlenit do sebe všechny dobré nápady ostatních frameworků. Struts 2 převyšuje Struts 1 co se týče funkcionalit a jejich implementace téměř po všech stránkách.

Nicméně stále platí, že daleko rozšířenější je původní verze, možná i pro menší množství softwarových nástrojů pro verzi 2 a jeho slabší podporu v dokumentaci.

2.9.1.1 Základní technologické výhody Struts 2 oproti Struts 1

Základní výhodou Struts 2 oproti Struts 1 jsou zejména:

- Implementace IoC (Inversion of Control): Tato technologie byla přejata z rámce Spring. Technologie bude vysvětlena v rámci této kapitoly u rámce Spring. Hlavním výsledkem aplikace IoC je zjednodušení akcí.

- Menší produkce objektů při získávání hodnot z formulářů (zrušení ActionFormů - ve Struts 1 to je speciální rozhraní, které musela rozšiřovat třída, jež vznikala právě pro získávání hodnot z formulářů na straně uživatele).

- Podpora jazyka OGNL - Ve Struts 2 lze pro výrazy použít jazyk OGNL (Object Graph Notation Language). OGNL je jazyk, pomocí něhož se dá definovat vazba mezi atributy tříd modelu a komponentami na straně uživatele (např. HTML elementy formuláře). Ze srovnávaných frameworků v rámci této práce využívá OGNL i rámec Tapestry. Další výhodou je skutečnost, že je OGNL ve Struts 2 využito pro typovou konverzi, což přináší výhodu využití typové konverze pro primitivní i objektové datové typy.

- Lepší volení implicitních hodnot parametrů – vede ke zestručnění kódu, jelikož v případě vhodně zvolené hodnoty ji není třeba již explicitně uvádět.

- Lepší tagy - ve Struts 2 generuje značku šablona asociovaná s touto značkou. Jedná se o analogii kaskádových stylů (css), kdy opakující se značky jednoduše přesuneme do šablony, čímž může dojít k výrazné úspoře kódu.

- Podpora anotací - jedná se o anotace zavedené jazykem Java ve verzi 5.

- Podpora integrace Struts2 a Springu - Struts2 zná JavaBeany Springu.

- Jednodušší vytváření pluginů - stačí vytvořit JAR archiv a připojit ho k aplikaci.

- Podpora technologie AJAX.

Podrobnější informace o rozdílech mezi oběma verzemi Struts a zejména o způsobu přechodu na verzi 2 budou probrány v kapitole 3 a 4.

Software potřebný k použití rámce:

- Vlastní Framework Struts (v dubnu 2011 k dispozici verze 1.3.10 ze 04.12.08 resp. verze 2.2.1.1 z 20.12.10).
- Java Development Kit (v dubnu 2011 k dispozici JDK 6 Update 21).
- JSP kontejner (např. Apache Tomcat, aktuálně verze 7.0.12).
- Nástroj pro kompilaci (např. Apache Ant, aktuálně verze 1.8.2).
- Ovladač pro přístup k databázi ovladač (JDBC).

2.9.1.2 Hodnocení Struts podle kritérií

Hodnocení rámce Struts dle určených kritérií je následující:

1) **Druh rámce**:

Požadavkem řízený.

2) **Ověření dat**:

a) Struts 1:

- Validace na straně uživatele: ano, prostřednictvím rámce Commons Validator, který lze editovat konfiguračním souborem web.xml.

- Validace na straně serveru: ano, prostřednictvím podtříd *ValidatorForm* a *ValidatorActionForm* třídy *ActionForm*. Tyto třídy jsou prostřednictvím kontextu použitelné i v jiných objektech, (jiné Action objekty, JSP). Framework disponuje sdíleným mechanismem upozornění na chyby a jejich následným zobrazováním.

Problematická je však situace, kdy jedno z API použitých při tvorbě aplikace prošlo nějakým významnějším vývojem. Potom většinou nezbývá než celou validaci přepsat od začátku.

b) Struts 2:

Validační funkcionalita je přímo komponenta jádra rámce, jež byla vyvinuta z *OpenSymphony*. Tato komponenta nabízí všechny funkcionality Commnons Validator Struts 1, ale mnohem jednodušší formou.

3) **Ztvárnění prezentační vrstvy**:

K dispozici je škála prezentačních systémů jako je standardní Java Server Pages (JSP včetně JSTL), šablonovací technologie Velocity Templates či XSLT.

4) **Internacionalizace-lokalizace**:
Struts 1 i Struts 2 převádí nativní podporu lokalizace z Java tříd *java.util.ListResourceBundle* a *java.util.PropertyResourceBundle*. Rozlišnost obou verzí spočívá až ve větší flexibilitě při výběru lokalizovaného svazku („language bundle").

a) Struts 1 - Rámec očekává vše pro překlad stránek v jediném souboru, např. se jménem *Bundle_fra.properties* či ve více souborech. Tento postup vyžaduje uživatelskou Java třídu navíc a potřebu prefixovat všechny zprávy značkou souboru tak, aby tato třída starající se o řízení překladu stránek mohla určit, ze kterého souboru má příslušný text získat. To je velmi neobratný přístup plodící chyby. Potřebné konstrukce pro internacionalizaci jsou volány v souboru *struts-config.xml* prostřednictvím elementu *message-resources*.

b) Struts 2 - Dává možnost variabilního rozsahu nastavení lokalizace počínaje určením souboru pro celou aplikaci (totožné se Struts 1) a konče elementární úrovní jediné akce.

5) **Míra samostatné testovatelnosti aplikace**:
a) Struts 1 - Pomocí souboru dostupných JUnit testů a Mock objektů (objekt podobný Skeletonu, jde vlastně o proxy objekt, který je schopen po volání svých metod vrátit předem nastavené hodnoty pro dané metody – z toho vyplývá možnost testovat i stavy závisející na kontejneru). Tyto postupy usnadňují integrační testování i testování tříd. Existují také speciální StrutsTestCase (speciálně vyvinutý balíček testů).

b) Struts 2 – Testování aplikace je jednodušší, již není třeba Mock objektů.

6) **Bookmarks-friendly**:
Ano. – V struct-config.xml (Struts 1), resp. struts.xml (Struts 2) se mapují všechny akce.

7) **Reakce na stisk specifických tlačítek prohlížeče**:
Řešitelné prostřednictvím akce *redirectAction* a vypnutím cache u uživatele posláním HTTP hlaviček:

pragma: no-cache
expires: -1
cache-control: no-cache

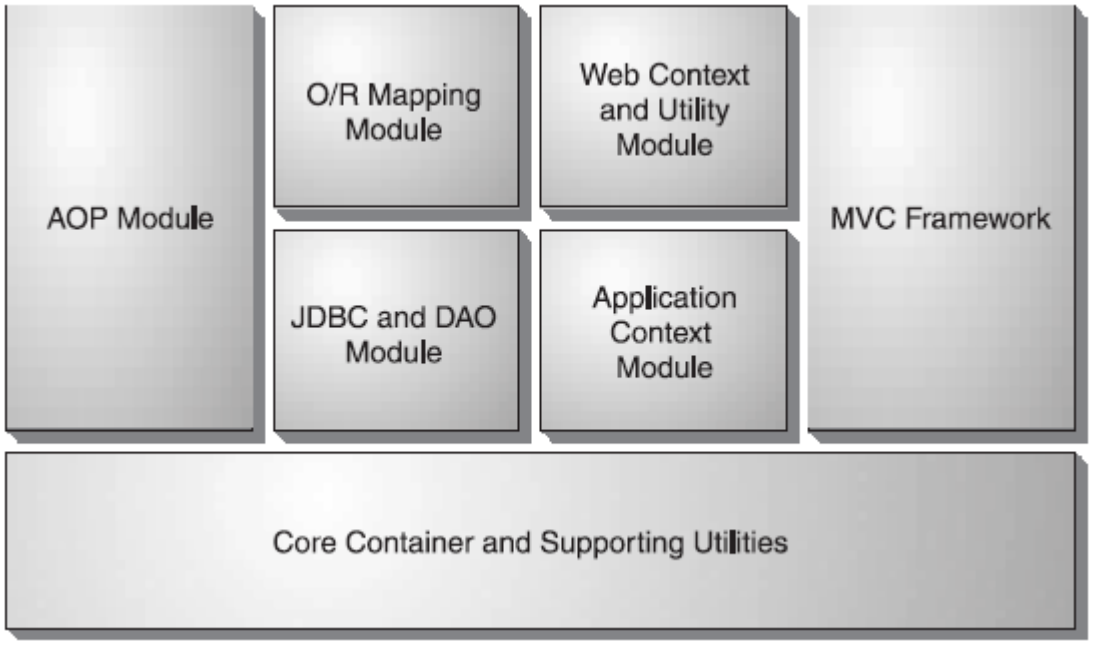
Obě verze tohoto rámce budou detailněji srovnávány v kapitole 3 a 4, proto je rámec v této kapitole probrán o něco stručněji než ostatní rámce. **Hodnocení silných a slabých stránek** rámce Struts v rámci porovnání Struts 1 a Struts 2 je dostatečně rozebráno v kapitole 3, na kterou tímto autor odkazuje.

2.9.2 Spring MVC

Domovský web: http://www.springsource.org/

Autorem je Rod Johnson, který jej v roce 2003 publikoval jako open-source kód. Původní návrh vznikl jako zefektivnění řešení banálních problémů, se kterými se programátor Java EE aplikací potkává a rovněž z důvodu snížení ceny i nákladů. Jde tedy o typické důvody pro vznik webového rámce.

Rámec Spring je tvořen sedmi moduly (viz obrázek 2.7). Jak je patrné, modul Spring MVC je součástí této sedmičlenné rodiny. Jako celek tyto moduly poskytují vše, co je potřeba k rozvoji robustních webových aplikací. Není ovšem nutné používat všechny moduly, které rámec nabízí. Existuje možnost vybrat si vlastní moduly mimo Spring a zbytek nepoužitých Spring modulů ignorovat. Jak je patrné z obrázku, stojí všechny moduly na „core containeru“. Tento kontejner definuje, jak jsou jednotlivé moduly vytvářeny, konfigurovány a řízeny.



Obr. 2.7: Modulární složení rámce Spring (WALLS C., 2005)

2.9.2.1 Moduly rámce Spring

The Core container

Poskytuje základní funkčnost rámce. V tomto modulu lze najít *BeanFactory*, srdce každé aplikace založené na Springu. *BeanFactory* odděluje konfigurační specifikace a závislosti aplikace od vlastního kódu.

Application context module

Tento modul dělá ze Springu rámec. Rozšiřuje *BeanFactory* přidáním podpory pro internacionalizaci, validaci a životní cyklus událostí aplikace. Navíc tento modul poskytuje další služby, jako je e-mail, JNDI přístup, EJB či vzdálenou komunikaci. Součástí je také podpora pro integraci se šablonovacími („templating") rámci jako jsou Velocity a FreeMarker.

Spring's AOP module

Modul AOP nabízí bohatou podporu pro aspektově-orientované programování. Tento modul je schopen zajistit vývoj vlastních aspektů. Pro zajištění interoperability mezi Springem a dalšími AOP rámci, je zajištěno rozhraní API definovaném AOP aliance. Tato aliance je založena právě pro účely sdílení funkcionality mezi různými rámci.

Podpora aspektově orientovaného programování (**AOP**) je vedle technologie Inversion of Control (IoC), jež bude popsána dále, jednou z **klíčových funkcionalit** celého Spring frameworku. AOP však již jde mimo rámec této práce, proto je zde alespoň tato zmínka.

AOP modul rovněž zavádí metadata programování do rámce. Použitím této funkcionality je rámec informován, kde a jak má použít aspekty.

JDBC abstraction a modul DAO

Práce s JDBC často vede k vytváření stereotypního kódu (získání connection, vytváření statementu, resultsetu, zavírání connection). Tento modul překrývá tento kód, takže vlastní kód tvořený programátorem a týkající se vlastní práce s databázi je kratší a čistější.

Kromě toho, tento modul používá AOP modul, který pak poskytuje řídící služby transakcí pro objekty ve Spring aplikaci.

Object/relational mapping integration module (ORM modul)

Tento modul je určen pro vývojáře, kteří dávají přednost objektově-relačnímu mapování (ORM) před JDBC. Spring zde nenabízí vlastní ORM řešení, ale poskytuje technologie jiných populárních ORM rámců jako jsou Hibernate JDO a iBATIS SQL Maps (v červnu 2010 vývoj ukončen). Spring poskytuje stejnou úroveň řízení každého z těchto ORM rámců jako v případě JDBC (WALLS C., 2005).

Spring's web module

Jedná se o kontext modul, který buduje v aplikaci kontext vhodný pro web aplikace. Kromě toho poskytuje podporu pro několik web-orientovaných funkcionalit jako je manipulace s požadavky více uživatelů na upload souborů a navázání parametrů požadavku na business objekty. V neposlední řadě nabízí také podporu integrace s rámcem Apache Struts.

Spring MVC framework

Jedná se o samotný MVC framework. Spring MVC lze snadno integrovat s ostatními MVC rámci jako je Struts (pomocí svého web modulu). Spring MVC rámec nabízí použití IoC („Inversion of Control") k poskytnutí přesného oddělení řídící vrstvy od business logiky. Samozřejmostí je napojení parametrů požadavku na business objekty. Spring může využít některou z dalších služeb Spring jako je zasílání zpráv internacionalizace („i18n") a validace.

2.9.2.2 Inversion of Control

Na tomto principu pracuje většina moderních komponentně orientovaných rámců, včetně níže uváděného Tapestry.

Pracuje na známém Hollywoodském principu – „Don't call us, we'll call you".

Poskytuje dva principy:

1) **Dependency injection**

Při komponentně orientovaném programování tvoří programátor množství komponent. Aby byly tyto komponenty jednoduše odstranitelné z aplikace a také aby mohly komunikovat s jinými komponentami, musí implementovat společná rozhraní.

Ve Springu je implementován IoC kontejner, který je vlastně mozkem aplikace a slouží právě ke shromažďování informací o jednotlivých komponentách aplikace, o rozhraních, která každá konkrétní komponenta implementuje. IoC kontejner je tedy delegován k řízení těchto

závislých objektů. Udržuje graf objektů, takže pro něj není problém na vyžádání podat informace o závislostech každé komponenty a poskytuje tedy objektům životní cyklus.

2) Druhou službou poskytovanou IoC kontejnerem je **řízení životnosti komponent**. Tato funkcionalita v podstatě znamená, že programátor nemusí v případě potřeby vytvářet jedináčky („singleton"), ale jednoduše si o ně řekne IoC kontejneru a ten se o poskytnutí postará.

2.9.2.3 Hodnocení Spring MVC podle kritérií

Hodnocení rámce Spring MVC dle určených kritérií je následující:

1) **Druh rámce**:
Požadavkem řízený.

2) **Ověření dat**:
Ano. - Slouží k němu rozhraní *org.springframework.validation.Validator*, které je definováno takto:

```
public interface Validator {
void validate(Object o, Errors e);
boolean supports(Class class);
}
```

Implementace tohoto rozhraní by měla prozkoumat pole objektu předaného metodě *validate()* a odmítnout neplatné hodnoty přes objekt *Errors*. Metoda *support()* pomáhá Springu stanovit zda je či není validátor vhodný pro předané třídy. Dále je třeba vytvořit vlastní třídu, např. *MyValidator*, která implementuje rozhraní *Validator* a která bude sloužit k vlastní validaci objektů.

3) **Ztvárnění prezentační vrstvy**:

Vrstva View ve Springu je obyčejný JavaBean. Z technologií je nejvyužívanější JavaServer Pages (JSP). Alterativou mohou být FreeMarker šablony nebo dokonce pohledy produkující PDF či Microsoft Excel dokumenty.

4) **Internacionalizace-lokalizace**:
Internacionalizace a následná lokalizace aplikace je řešena použitím různých pohledů pro konkrétního uživatele. Děje se tak pomocí view resolver.

View reslover je jakýkoliv Bean, který implementuje *org.springframework.web.servlet. ViewResolver*. Spring MVC takový Bean pozná a konzultuje s ním, kterou vrstvu Pohled má použít. Spring disponuje těmito 4 implementacemi ViewResolver: *InternalResourceViewerResolver*, *BeanNameViewResolver*, *ResourceBundleViewResolver*, *XmlViewResolver* pro převod logických jmen pohledů na potřebné objekty.

5) **Míra samostatné testovatelnosti aplikace**:

Podpora pro testování existuje v spring-mock.jar. Testy se neomezují pouze na správné psaní kódu webového rámce Spring, ale jsou i schopny testovat správnost např. SQL příkazů v rámci datového přístupu JDBC. Testy se spouští uvnitř IDE, není třeba je nasadit do aplikačního serveru.

6) **Bookmarks-friendly**:

Ano, bezchybně.

7) **Reakce na stisk specifických tlačítek prohlížeče**:

Ano. - Pomocí metody *getTargetPage()*, která odstraňuje z konkrétního parametru požadavku stránky „_target“ prefix a nahrazuje jej číslem. Soupis těchto čísel je veden v seznamu stránek („pages list“).

2.9.2.4 Shrnutí silných a slabých stránek rámce Spring MVC

Silné stránky rámce:

- Podpora moderních technologií, přístupů podporujících efektivnější tvorbu web aplikací jako je: AJAX, OGNL, Dependency Injection. Špičková je zejména podpora technologie **AJAX**.
- Testování aplikace je jednoduché, jde dokonce nad rámec Javy možností testovat SQL příkazy v rámci přístupu přes JDBC.
- Dokumentace na vysoké úrovni, srovnatelné s JavaServer Faces.
- Osvojení si rámce programátorem není příliš časově náročné.
- Solidní validace dat na straně uživatele i serveru.

Slabé stránky rámce:

- Pro prezentační vrstvu je standardem JSP, zde je konkurence již dál, např. s JSF 2.0 Facelets.
- Produktivita práce průměrná, ne však slabá.
- Minimální podpora zásuvných modulů (pluginů).

2.9.3 Apache Tapestry

Domovský web: http://tapestry.apache.org/

Autorem rámce Tapestry je Howard Lewis Ship, jež ho vytvořil jako vlastní nezávislý projekt. Později byl přijat do rodiny projektů Apache. Hlavním rozdílem oproti jiným rámcům je skutečnost, že nepoužívá technologii JSP, ale je postaven přímo nad technologií Servlet API. Prezentační vrstvu zajišťuje vlastním systémem HTML šablon, do kterých vkládá Tapestry a OGNL značky. Problémem Tapestry je ovšem skutečnost, že **každá nová generace je silně zpětně nekompatibilní** (momentálně jde o generace 4 a 5), tudíž přechod vývojáře od jedné ke druhé znamená v podstatě naučit se zcela nový rámec. Hlavní novinkou verze 5 je technika psaní kódu metodou zdola nahoru. Jak bylo již dříve řečeno, Tapestry patří spolu s rámcem JSF mezi ty, které podporují vyšší míru abstrakce nad protokolem HTTP a stejně jako JSF je zaměřen hlavně na vrstvu Pohled.

Aktuální verze rámce Apache Tapestry v dubnu 2011 je 5.2.5.

2.9.3.1 Životní cyklus Tapestry

Životní cyklus rámce Tapestry se odehrává v těchto krocích:

1. HTTP požadavek je přijat servletem Tapestry k vyřízení.

2. Požadavek je servletem převeden na událost a řízení je servletem předáno korespondujícímu objektu stránky či rovnou její metodě. Objekt stránky je třídou rozšiřující vnitřní třídu *BasePage* či *AbstractPage*. Typický objekt stránky obsahuje Getter i Setter metody pro objekty dané stránky. Metody jsou vyvolávány akcí uživatele na svém rozhraní v různých stavech životních cyklů metody i stránky.

3. Korespondující metoda třídy stránky je vyvolána podle requestu, tato metoda již má svůj životní cyklus.

2.9.3.2 Hodnocení Tapestry podle kritérií

Hodnocení rámce Tapestry dle určených kritérií je následující:

1) **Druh rámce**:

Komponentně orientovaný.

2) **Ověření dat**:

Ano. - prostřednictvím rozhraní *Validator* (slouží k tomu 2 parametry *validators* a *displayName*). Validace jsou podporovány na straně uživatele i serveru.

3) **Ztvárnění prezentační vrstvy**:

Typicky vlastním systémem HTML šablon, do kterých vkládá Tapestry značky. Obecně nemusí jít jenom o HTML, ale jakýkoliv validní značkovací jazyk.

4) **Internacionalizace-lokalizace**:

Řešeno s pomocí javovských *Locale* a *ResourceBundle* (podobně jako Struts).

5) **Míra samostatné testovatelnosti aplikace**:

Testování bez nasazení („deploy") webového kontejneru je možné, ale protože je nutné rozšířit základní třídy rámce, není testování tak pohodlné jako například ve frameworku Java Server Faces. Slabší podpora testovatelnosti aplikace je jednou z klíčových nevýhod rámce.

6) **Bookmarks-friendly**:

Ano. – ovšem ne právě jednoduše prostřednictvím komponenty *IExternalPage*.

7) **Reakce na stisk specifických tlačítek prohlížeče**:

Ano. – řešitelné pomocí výjimky *RedirectException*, jež je do Tapestry zakomponována.

2.9.3.3 Shrnutí silných a slabých stránek rámce Tapestry

Silné stránky rámce:

- Tvorba nových komponent je velmi jednoduchá.
- Dobře vymyšlená a výborně zpracovaná validace dat na straně uživatele i serveru.
- Podpora moderních technologií, přístupů podporujících efektivnější tvorbu web aplikací jako je: AJAX, OGNL, Dependency Injection.
- Celková efektivita práce po náročnějším osvojení si rámce programátorem je výborná.

Slabé stránky rámce:

- Zásadním nedostatkem je odlišnost a velká **zpětná nekompatibilita** nových verzí rámce oproti starým. Tato skutečnost vede samozřejmě k menší oblíbenosti mezi zadavateli webových aplikací, v důsledku čehož existuje i menší ochota vývojářů používat tento rámec.
- První nedostatek ještě zvýrazňuje **horší podpora testovatelnosti** aplikace.
- Podpora při vývoji v hlavních Java prostředích (NetBeans a Eclipse) je pouze ve formě zásuvných modulů (pluginů).
- Tvorba komponent je sice silnou stránkou rámce, ovšem technika použití těchto komponent je již relativně složitá.
- Horší kvalita dokumentace vedoucí k pomalejšímu osvojení si rámce vývojářem.

2.9.4 JavaServer Faces

Domovský web: www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html.

Aktuální verze rámce JavaServer Faces v dubnu 2011 je 2.1.1.

JSF je komponentně orientovaným rámcem pro spíše uživatelské rozhraní aplikace.

Je nutné rozlišit verze, kdy počáteční 1.0 a 1.1 (rok 2004) byly kritizovány zejména ze dvou důvodů: komplikované až těžkopádné použití a navigace založena pouze na metodě POST.
Zejména od aktuální verze 2.0 (rok 2009) jsou tyto problémy vyřešeny, tvorba již není tak těžkopádná a lze využít i metodu GET a funguje již plná podpora AJAX.
Způsob fungování JSF je dobře čitelný z následujícího životního cyklu frameworku.

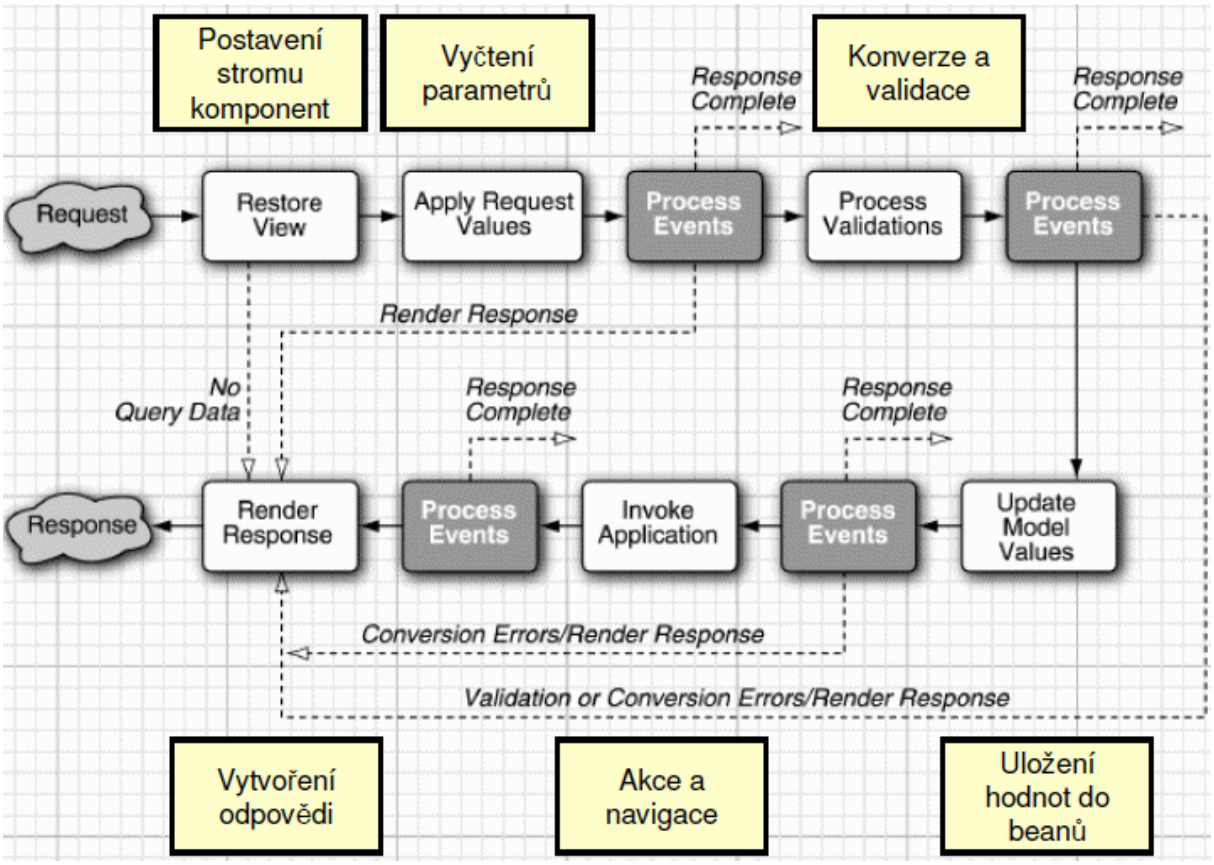
2.9.4.1 Životní cyklus JSF

Životní cyklus webové aplikace JSF začíná, když uživatel vytvoří požadavek. Viz obrázek 2.8.

1. Restore view phase (Obnovení pohledu):

Tato fáze začíná, když uživatel odešle požadavek („HTTP request") zobrazení nějaké JSF stránky. Dojde k napojení komponent na obsluhovače událostí („Event handlers") a validátory. Tento pohled, strom komponent („Component Tree") je uložen v objektu *FacesContext*. *FacesContext* ukládá tento pohled ve viewRoot property.

Požadavek přichází přes řadič FacesServlet, ten zkontroluje požadavek a ukládá jako ID pohledu („View ID") název stránky. View ID se používá k vyhledání komponentů v aktuálním zobrazení. JSF řadič používá toto číslo v případě, že pohled již existuje. Pokud pohled ještě neexistuje, JSF řadič jej vytváří.



Obr. 2.8: Životní cyklus JavaServer Faces (GEARY D., 2007)

2. Přiřazení požadavků (Apply request values phase):

Po obnovení stromu komponent v předchozí fázi každá komponenta ve stromu obdrží svou novou hodnotu a lokálně si ji uloží. Hodnoty komponent se typicky získávají z parametrů požadavku. Jakmile dojde k uložení hodnoty prvního atributu („immediate atribute") komponenty na „*true*", je spuštěn proces konverze a události spojené s komponentou. Pokud konverze selže, je vygenerována chybová zpráva asociovaná s komponentou a je zařazena do FacesContext. Na konci této fáze tedy komponenty mají nastaveny nové hodnoty. Zprávy a události jsou uloženy do FacesContext.

3. Proces validace (Proces validations phase):

Během této fáze jsou hodnoty uložené ve stromu komponent porovnány s ověřovacími pravidly registrovanými pro každou komponentu. Neprojde-li hodnota ověřením úspěšně, je do FacesContextu uložena chybová zpráva a komponenta je považována za neplatnou. Životní cyklus takové neplatné komponenty poté skočí až do poslední fáze vytvoření odpovědi („Render response phase") a zobrazí hlášení o validační chybě.

4. Aktualizace hodnot modelu fáze (Update model values phase):

Pokud proběhne úspěšně předchozí fáze, místní hodnoty komponent mohou být zapsány do svých odpovídajících objektů na straně serveru, jimiž jsou beany („backing beans"). Pokud místní hodnoty komponent nelze převést na typy určené vlastnostmi beanu („bean properties"), skáče životní cyklus komponenty opět přímo na poslední fázi a je zobrazeno chybové hlášení.

5. Zavolání aplikace (Invoke application phase):

Před touto fází byly hodnoty komponent převedeny, ověřeny a zapsány do objektů beanů svých protějšků. Takže nyní mohou být použity k aplikaci business logiky. Tento kód na úrovni aplikace se provádí například odesláním formuláře či odkazem na jinou stránku.

6. Vytvoření odpovědí (Render response phase):

V této fázi JSP kontejner vytváří odpověď uživateli. Postupně jsou volány z FacesServletu hodnoty jednotlivých komponent potřebných k odpovědi. Pokud je tato fáze spuštěna po neúspěšném zakončení přiřazovací, validační či aktualizační fáze, měla by aplikace skočit na první stránku se zobrazením chybového hlášení (RoseIndia.Net, 2008).

2.9.4.2 Typické složení JSF komponenty

Komponenta rámce JSF má většinou následující složení:

- UIComponent class - Třída obsahuje business logiku. Její instance tvoří části stromu komponent.
- Renderer class - Třída převádí komponenty do značkovacího jazyka (typicky HTML). Ve specifických případech může její funkci převzít UIComponent třída.
- Tag class - Předává parametry do třídy komponenty, slouží pro odkaz v JSP stránce na komponentu.

- Tag Library Descriptor (TLD) - XML soubor popisující vlastnosti a umístění Tag class. Rámec si nejdříve volá TLD, aby zjistil, kde se nachází a jak funguje Tag class.

2.9.4.3 Hodnocení JSF podle kritérií

Hodnocení rámce JavaServer Faces dle určených kritérií je následující:

1) **Druh rámce**:
Komponentně orientovaný.

2) **Ověření dat**:

Ano. - Proces validace dat se děje ve třetí fázi životního cyklu JSF aplikace, bezprostředně jí předchází fáze přiřazení hodnot z parametru požadavku, součástí které je konverze dat.

- **Konverze dat**:

Konvertory čísel a dat jsou možné na vstupu (funkce parseru) i výstupu (funkce formátování do výstupní podoby).

U vstupů vázaných na primitivní datové typy není třeba konvertory specifikovat, jsou spuštěny automaticky. Naopak pro konverzi datumu platí nutnost použití externího konvertoru. Definice konverze se provádí faces značkami *<f:convertNumber>*, *<f:convertDateTime>* nebo jako atribut converter značky *<h:inputText>*. Za atribut converter se pak dosazují konvertory *javax.faces.DateTime*, *javax.faces.Long* apod.

Vývojář také může napsat vlastní konvertor, jenž je nutné zaregistrovat v *faces-config.xml*. Následně je možné jej použít v atributu converter značky *<h:inputText>* způsobem, jaký je popsán výše. Druhou možností je využít značku faces *<f:converter>*. Třetí možností je napojení hodnoty výrazu („value binding expression"). Výhodou je přístup konvertoru k proměnným beanu. Metoda musí být typu get a musí vracet *javax.faces.convert.Converter*.

- **Validace dat:**

Validátor je třída implementující rozhraní *javax.faces.validator.Validator*, která obsahuje metodu void *validate(FacesContext context, UIComponent component, Object value)*.

Pokud se při konverzi objeví chyba, generuje se chybové hlášení *javax.faces.application.FacesMessage* a to se vyhodí s výjimkou *javax.faces.validator.ValidatorException*. Uživateli je poté zobrazena stránka, na které se vyskytla chyba. Hlášení jsou zobrazena ve značce *<h:message>*.

Vestavěná validace používá faces značek *<f:validateLength>*, *<f:validateLongRange>* atp., u kterých se nastaví atributy minimum a maximum.

Vlastní validace (analogie s vlastním konvertorem) je také možná. Validátor je nutné registrovat v *faces-config.xml*. Poté je možné použít značku *<f:validator>*. Další možností je napojení hodnoty výrazu analogicky jako u konverze dat. Výhodou je přístup validátoru k proměnným beanu. Metoda musí mít stejné parametry jako metoda *validate()* rozhraní *javax.faces.validator.Validator*

3) **Ztvárnění prezentační vrstvy**:
Na počátku byla snaha využít pro vrstvu View technologii JSP, jelikož však obě technologie mají velmi rozdílný životní cyklus (JSF daleko složitější), bylo potřeba vyvinout jinou technologii, která by byla schopna využít veškerý potencionál JSF. Byla tedy vyvinuta technologie Facelets, která je standardní součástí JSF 2.0. Kromě toho, že je schopna využít plně potenciál JSP, je vyvinuta na bázi XML, což je pro většinu programátorů známá technologie. Takže pro efektivní rychlou tvorbu v JSF 2.0 stačí místo případného zdlouhavého učení se JSP osvojit si několik XML tagů.

4) **Internacionalizace-lokalizace**:
Řešeno s pomocí javovských *Locale* a *ResourceBundle* (podobně jako Struts).

5) **Míra samostatné testovatelnosti aplikace**:
Pomocí Mock objektů. Ty usnadňují integrační testování i testování tříd.

Spousta funkcionalit JSF (např. internacionalizace a testovatelnost aplikace) je velmi podobně či dokonce totožně implementovaná jako v rámci Struts. Důvodem je osoba Craiga McClanahana, který je autorem rámce Struts a původním spoluvedoucím projektu JSF.

6) **Bookmarks-friendly**:
JSF do verze 1.1 – Ne, neboť podporuje pouze metodu POST protokolu HTTP.
JSF verze 2.0 – Ano, neboť podporuje metodu GET protokolu HTTP.

7) **Reakce na stisk specifických tlačítek prohlížeče**:
JSF do verze 1.1 – Není standardně zahrnuto.

2.9.4.4 Shrnutí silných a slabých stránek rámce JSF

Před hodnocením silných a slabých stránek rámce je nutné na tomto místě nejprve zhodnotit výhody tohoto rámce ve verzi 2.0 před svým předchůdcem.

Výhody JSF 2.0 před JSF 1.1:

- Efektivnější tvorba prezentační vrstvy prostřednictvím technologie **Facelets**, která stojí na technologii XML. Tuto skutečnost většina programátorů ocení, hlavně v kontrastu se zdlouhavějším a těžkopádnějším řešením JSF první generace, která využívala pro prezentaci Java Server Pages (JSP).
- Podpora moderních technologií a postupů jako je AJAX, Dependency Injection.
- Podpora metody GET požadavku HTTP zaručující schopnost aplikací adresovat každou svou stránku záložkou („bookmark“).

Silné stránky JSF 2.0:

- Součást prostředí Java EE, z čehož plynou tyto výhody:
  - Dostatek dokumentace.
  - Široká podpora firem tvořících vývojové nástroje pro Javu (Eclipse, NetBeans).
  - Firmy se neobávají ztráty z investic do vzdělání zaměstnanců (z hlediska JSF).
  - Velká „členská základna“ programátorů.
- Efektivní tvorba prezentační vrstvy prostřednictvím technologie **Facelets**.
- Podpora moderních technologií a postupů jako je AJAX, Dependency Injection.
- Rychlost osvojení si práce s rámcem (oproti např. Tapestry).

Slabé stránky JSF 2.0:

- Produktivita práce v porovnání s např. Tapestry se zdá nižší.
- Rychlost tvorby nových komponent a jejich následné využití.

2.9.5 Hodnocení vybraných rámců z hlediska praxe

Na závěr kapitoly hodnotící čtyři vybrané frameworky budou uvedeny tabulka a graf z praxe, vypovídající o těchto rámcích z pozic vývojářů, respektive poptávky po nich na trhu práce.

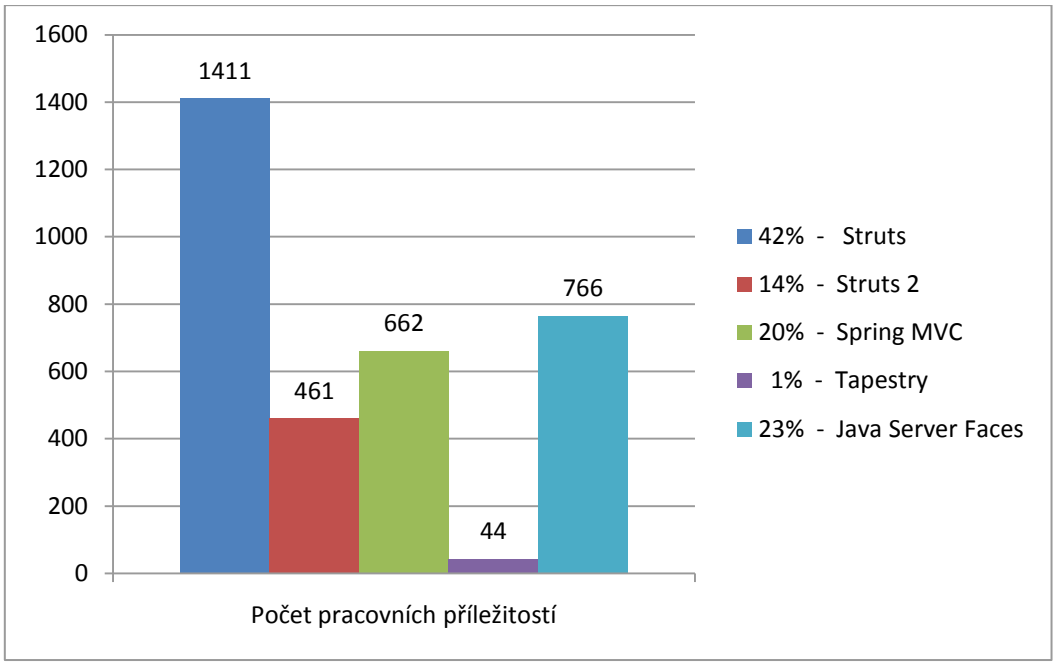
První tabulka (viz tabulka 2.2) zobrazuje subjektivní hodnocení jednotlivých frameworků vývojáři firmy Raible Designs, která se specializuje na vývoj a zlepšování webových aplikací. Každému kritériu je přiřazena váha („weight“) z množiny 0, 5 a 10 (více je lépe). Každý rámec byl ohodnocen v každém kritériu, přičemž dostal hodnocení rovnající se maximálně

váze daného kritéria (v tom případě je rámec hodnocen v daném kritériu 100%). Vítězem se stal rámec Spring MVC, který dosáhl 85% výsledku. Pozice JSF je zkreslena nezahrnutím její novější verze 2.0, nýbrž starší 1.2. Stejně tak tomu je v případě rámce Struts (zde starší verze 1).

Weight	Criteria	Struts	Spring MVC	JSF v1.2	Tapestry
10	Developer Productivity	5.00	5.00	5.00	10.00
0	Developer Perception	0.00	0.00	0.00	0.00
5	Learning Curve	5.00	5.00	2.50	2.50
5	Project Health	2.50	5.00	5.00	2.50
5	Developer Availability	2.50	5.00	5.00	5.00
0	Job Trends	0.00	0.00	0.00	0.00
0	Templating	0.00	0.00	0.00	0.00
5	Ajax	2.50	5.00	2.50	2.50
5	Plugins or Add-Ons	2.50	0.00	5.00	2.50
10	Scalability	10.00	10.00	5.00	5.00
10	Testing	10.00	10.00	5.00	10.00
0	i18n and l10n	0.00	0.00	0.00	0.00
5	Validation	5.00	5.00	2.50	5.00
10	Multi-language Support	5.00	5.00	10.00	10.00
10	Quality of Documentation/Tutorials	5.00	10.00	5.00	5.00
0	Books Published	0.00	0.00	0.00	0.00
10	REST Support (client and server)	5.00	10.00	0.00	5.00
10	Mobile / iPhone Support	10.00	10.00	10.00	10.00
0	Degree of Risk	0.00	0.00	0.00	0.00
100	**Weighted Totals**	**70**	**85**	**62.5**	**75**

Tab. 2.2 Srovnání webových rámců dle různých kritérií (Raible, 2011)

Následující graf 2.1 ukazuje, jaká je v USA poptávka po vývojářích sledovaných pěti webových rámců. Dá se říci, že trh stále ovládá požadavkem řízený framework Struts. S určitým odstupem, který se v poslední době neustále zmenšuje, se umístily modernější Spring MVC a JavaServer Faces. Hodnoty byly získány 25. dubna 2011 na serveru www.dice.com, což je přední americký server zabývající se nabídkou zaměstnání v technických oborech.



Graf 2.1: Postavení vybraných rámců na pracovním trhu (Dice Holdings, 2011)

3 Analýza stávajícího využití webových rámců ve firmě

Tato kapitola se zabývá analýzou stávajícího využití webových rámců ve firmě CompuGroup Medical Česká republika s.r.o., Divize Nemocniční informační systémy, Smetanova 9, 602 00 Brno. Jednou z činností firmy je implementace nemocničního informačního systému CGM G3 Systema.

CGM G3 Systema představuje provozní řešení pro nemocnice, které podporuje celý proces léčby pacienta, a to od jeho přijetí přes stanovení diagnózy a terapeutické procesy až po propuštění pacienta z nemocnice a vyúčtování poskytnuté zdravotní péče. Systém CGM G3 Systema je propojený s jednotkou workflow a nabízí zákazníkovi možnost navrhnout řešení, které vyhovuje procesům v jeho organizaci a které lze těmto procesům přizpůsobit.

Stručná charakteristika systému

CGM G3 Systema poskytuje řadu funkcí, jako je správa dat pacienta, lékařská a sesterská dokumentace, správa objednávání, systém zpráv a vyúčtování pacienta. Systém CGM G3 Systema je prvním integrovaným NIS, který podporuje všechny běžné lékařské a sesterské procesy a díky širokým možnostem nastavení a přizpůsobení tak poskytuje účinné nástroje nejen pro vedení veškerých potřebných agend, ale také pro podporu procesního řízení nemocnice.

CGM G3 Systema disponuje veškerými potřebnými funkcemi pro podporu každodenních rutinních postupů typických pro nemocnice všech velikostí, a to od příjmu nebo vytvoření žádanky až po stanovení diagnózy, vytvoření pacientského záznamu a vyúčtování poskytnuté péče. CGM G3 Systema je silným nástrojem pro automatizaci procesů, čímž významně podporuje klinickou práci zdravotnického personálu a volbu správných lékařských postupů a služeb (SHERWOOD, 2011).

Typický projekt implementace pak má toto zadání:

Projekt Implementace nemocničního informačního systému (dále NIS) do ÚVN Praha:

CGM funguje jako subkontraktor, dodává pouze NIS (celá zakázka obsahuje více řešení, které již nespadají pod firmu CGM).

Cíle projektu jsou následující:

- Nahrazení současného terminálového řešení NIS (název AMIS, spol.ICZ) moderním celoevropským řešením (G2 a G3 moduly - G2 Java, G3 Adobe Flash technologie).

- Kompletní pokrytí oblastí řešenými NIS (administrativa pacientů, podpora medicínských procesů, plánování návštěv, vyúčtování pojišťovně, komunikace s komplementy (laboratoře, Radiologie apod.), komunikace s použitím standardů ve zdravotnictví (HL7, IHE , webové služby).

- Provázanost všech (většiny) SW používaných na pracovištích ÚVN prostřednictvím integrační platformy, tedy vytvoření centrálního systému schopného komunikace se všemi subsystémy.

V rámci třetího cíle (implementace integrační platformy) se na některá pracoviště integrují subsystémy běžící jako webová aplikace, kdy je z různých příčin nevhodná (či ne nezbytná) implementace NIS i na toto místo. Pro účely této práce je podstatný fakt, že v těchto subsystémech běží webová aplikace na platformě Java 2 Enterprise Edition a zejména firma používá v těchto aplikacích dominantně rámec Struts 1 (classic). Proto bude dalším předmětem práce otázka, zda je rámec vhodné zaměnit za modernější.

3.1 Zhodnocení způsobilosti dosavadního rámce Struts 1

3.1.1 Proč Struts 1 vyměnit?

Aspekty hovořící pro výměnu rámce Struts 1 jsou tyto:

a) Ukončení podpory

Organizace Apache Software Foundation se již několik let dává slyšet, že podpora Struts 1 bude ukončena. Napovídal by tomu fakt, že poslední nová verze Struts 1 pochází z prosince 2008. Proti hovoří skutečnost, že Struts 1 je celosvětově mezi vývojáři stále nejoblíbenější a disponuje silnou dokumentací. Nicméně architektonická zastaralost rámce je nezpochybnitelná, tvorba v modernějších rámcích je efektivnější.

b) Efektivnost tvorby

Modernější rámce jsou vytvářeny jako nástupci starších (tedy i Struts 1), takže přirozenou snahou jejich tvůrců je vyhnout se chybám svých předchůdců. V důsledku toho bývá práce s modernějším rámcem rychlejší, jednodušší a tedy efektivnější.

c) Nové technologie

S překotným vývojem zejména různých softwarových technologií dochází k tomu, že ty modernější a výkonnější využívají logicky pouze ty mladší frameworky. Zde je potřeba uvést již zmíněné technologie Inversion of Control, podporu jazyka OGNL či AJAXu, jež nesporně vedou k větší efektivnosti tvorby či prezentačních schopností aplikace. Ani jednu z těchto technologií Struts 1 neumí a v budoucnu ani umět nebude. Mohou sice nastat pokusy o jejich integraci, ale výsledek pravděpodobně zůstane za očekáváním, neboť Struts 1 bylo vyvíjeno v jiné době, kdy se s AJAXem či OGNL vůbec nepočítalo.

d) Integrace s jinými rámci

V době svého vzniku byl Struts 1 takříkajíc neomezeným vládcem mezi rámci, takže nebyl ani důvod implementovat mu schopnosti integrace s jinými rámci. V dnešní době, kdy je na scéně přece jen více úspěšných rámců, myslí jejich tvůrci při vývoji i na ostatní a snaží se vkládat do nich prvky zajišťující jejich vzájemnou snadnější integrovatelnost. Typicky jde např. o podporu rámce Springs ve Struts 2.

e) Výkonová zátěž

Jelikož novější rámce jsou nástupci Struts 1, bývá jejich řešení efektivnější i z hlediska potřeby vytvářet různé pomocné objekty či uskutečňovat více režijních akcí spojených s během aplikace.

f) Testování aplikace

Mladší rámce bývají lépe uzpůsobeny pro testování aplikace. Struts 1 vytváří například výše zmíněné Mock objekty, které u mladších rámců nenajdeme. Jejich tvorba přináší další nároky na výkon.

Všechny tyto argumenty jsou z hlediska firmy relevantní. Zejména větší efektivnost tvorby s modernějším rámcem a možnost efektivnějšího testování je pro firmu zásadní. Případná úspěšná změna frameworku na modernější přinese v každém případě časové či finanční úspory. A rovněž hrozba ukončení podpory je znepokojující.

3.1.2 Proč Struts 1 nevyměnit?

Aspekty hovořící pro výměnu rámce Struts 1 jsou tyto:

a) Hrozba neúspěšného přechodu

Míra schopnosti relativně jednoduše a bezbolestně přejít na práci s novým frameworkem je determinujícím aspektem úspěšného přechodu. Všichni se musí připravit na to, že v úvodní fázi přijde nepochybně pokles produktivity práce, který však musí být časově zvládnutelný a nesmí způsobit kolaps v práci vývojářů, kteří by v důsledku takového vývoje byli neúměrně stresováni nedodržením termínu svých úkolů. Je nutno počítat s počáteční nechutí vývojářů učit se něco nového, když s tím starým to šlo a jde přece tak dobře.

b) Tvorba dodatečných nákladů

Zde se řadí náklady na zaškolení a přeučení vývojářů, náklady na dodatečné testování přestavěných aplikací či náklady dodatečného odstraňování chyb, které převod aplikace na jiný framework přinese.

c) Dostatečnost dosavadního řešení

Funkčnost dosavadního řešení je v podstatě dostatečná a ve výhledu do budoucna se neočekává rozšiřování funkcionalit.

Největší hrozbou, která by mohla zastavit samotný přechod, je přehnaný důraz na šetření v tvorbě dodatečných nákladů. Pokud se firma už jednou pro transformaci rozhodne, neměla by zejména zbytečně šetřit na nákladech na zaškolení, neboť by se jí to nepochybně vymstilo zpomalením tvorby budoucích zakázek. Pokud by firma usoudila, že dosavadní řešení vyhovuje (ad c)), je třeba ji upozornit, že toto řešení je časovanou bombou, jelikož po ukončení podpory Struts 1 se může objevit potřeba funkcionality, která již nebude do rámce zahrnuta. Riziko je velké, neboť je předpoklad, že nemocnice budou fungovat v dlouhodobém horizontu.

Odpověď na otázku způsobilosti dalšího používání rámce Struts1 je záporná. Jednoznačně převyšují klady nad zápory. Tento poměr rizik a přínosů může reálně zvrátit pouze neúspěšný přechod na modernější rámec způsobený podceněním příprav (zejména jeho podfinancování).

4. Návrh případných změn ve využití frameworků ve firmě

4.1 Volba nástupnického rámce

Nejlepší volbou stát se nástupnickým rámcem pro řešení webových aplikací předestřených v kapitole 3 se jeví rámec **Apache Struts 2.**

Důvody volby:

Hlavním a zároveň jediným důvodem je příbuznost rámců Struts 1 a Struts 2:

a) Jelikož Struts 2 je stejně jako jeho předchůdce požadavkem řízený a je jeho nástupcem, bude nejjednodušším řešením pro přechod právě on. Jeho podobnost se Struts 1 bude nepochybně ze všech moderních rámců největší, což bude jistě přinášet úspory časové i finanční. A kritická úvodní doba přechodu bude méně problematická než u jakéhokoliv jiného frameworku.

b) Struts 2 obsahuje plugin, jež umožňuje použití akcí i formulářů ze Struts aplikace. To je důležité zejména pro již existující aplikace, které by se firma rozhodla převést.

c) Struts 2 drží v podpoře moderních technologií (AJAX, OGNL) a použitím moderních programátorských principů (Inversion of Control) krok se svou komponentně orientovanou konkurencí. A jelikož potřeby zadavatele webové aplikace nekladou žádné zvláštní nároky na prezentační stránku aplikace, není doména komponentně orientovaných rámců (síla v prezentaci výsledných dat) v tomto případě důležitá.

4.2 Základní implementační rozdíly rámců Struts 1 a Struts 2

Následující postup je důležitý v případě, že se firma rozhodne pro přechod na rámce Struts 2 i u již existujících webových aplikací běžících na některých pracovištích nemocnice. V případě tvorby nové aplikace je samozřejmě situace jednodušší, příslušný vývojář obeznámený s tvorbou pod Struts 2 napíše potřebnou aplikaci již čistě pod tímto rámcem.

4.2.1 Akce

Struts 2 je stejně jako Struts 1 založen na práci s akcemi („action"). Ale mechanismus akcí funguje trochu jinak. Nejprve je třeba pochopit, že akce už není jedináčkem.
Každý požadavek dostane svou vlastní instanci akce. To znamená, že lze dosáhnout instance proměnných složitých datových typů. Další změnou je, že akce se stává nezávislou na Servlet API.

Na obrázku 4.1 je zobrazena typická akce Struts 1. Lze vysledovat, že objekty Servlet API i Struts 1 jsou svázány v jediné části kódu. Nejenže to stěžuje spouštění akce při testování aplikace, ale také dochází ke stírání rozdělení odpovědnosti akce a serveru.

```
1   public class VzorovaAkce extends Action {
2
3       public ActionForward execute(
4               ActionMapping mapping,
5               ActionForm form,
6               HttpServletRequest request,
7               HttpServletResponse response)
8       {
9           VzorovyWebForm webForm = (VzorovyWebForm) form;
10          // zde bude business logika...
11          return mapping.findForward("success");
12      }
13  }
```

Obr. 4.1: Typická akce Struts 1

Třída *VzorovaAkce* musí být potomkem třídy Action (řádek 1).
Mapování (*ActionMapping*) a *ActionForm* (třída konvertující a validující hodnoty formuláře) musí obdržet metoda *execute()* (řádky 3,4,5).
Průchod formuláře metodou *execute()* je ve Struts 1 nestandardní. Musí se zapsat do podtřídy (řádek 9), která může vyhodit výjimku a je brzděn množstvím stringů z uživatelova vstupu, které se musí převést na Java datové typy.
Metoda *execute()* musí také přijmout odkazy pro požadavek a odpověď od Servlet API (řádky 6, 7).
Musí být také vytvořen objekt *ActionForward* a navrácen rámci Struts 1 (řádek 11).

Z tohoto popisu je zřejmé, že se tady děje příliš mnoho operací, které by mohl vykonávat framework sám. Na následujícím obrázku 4.2, je zobrazena odpovídající akce Struts 2.

```
1   public class VzorovaAkce{
2
3       private VzorovyBean model;
4
5       public String execute() {
6   // zde bude business logika...
7           return "success";
8       }
9       public VzorovyBean getModel() {
10          return model;
11      }
12      public void setModel(VzorovyBean model) {
13          this.model = model;
14      }
15  }
```

Obr. 4.2: Typická akce Struts 2

Třída *VzorovaAkce* ve Struts2 je jednoduchý POJO (řádek 3).
Třída už také není jedináčkem, ale existuje proměnná instance rezervovaná pro vstupy uživatele (řádek 3).
Metoda *execute()* již neočekává žádné argumenty a dokonce může mít libovolný název (řádek 5).
Návratová hodnota je obyčejný string, je to symbolické jméno, kterým rámec určuje, co se má stát dále (řádek 7).
A konečně *get/setModel()* slouží rámci k udržení interních hodnot zásobníku synchronizované se vstupy uživatele (řádek 8 až 12).

Jak je patrné z předchozího popisu, akce JavaBeans rámce **Struts 2** neobsahuje **žádné Servlet API závislosti**, takže testování je snadné. Skutečnost, že metoda *execute()* může být volána odkudkoliv, je pro programátora velmi užitečná.

4.2.2 Eliminace ActionForm ve Struts 2

ActionForm byl ve Struts 2 **vypuštěn**. *ActionForm* existoval v podstatě jen jako most pro kyvadlovou dopravu mezi konkrétní akcí a webovou stránkou. Na takovou jednoduchou funkci zabíral příliš mnoho systémových prostředků, proto bylo ve Struts 2 vymyšleno efektivnější řešení. Struts 2 je schopen velmi rychle aktualizovat model business logiky dle vstupů uživatele.

.

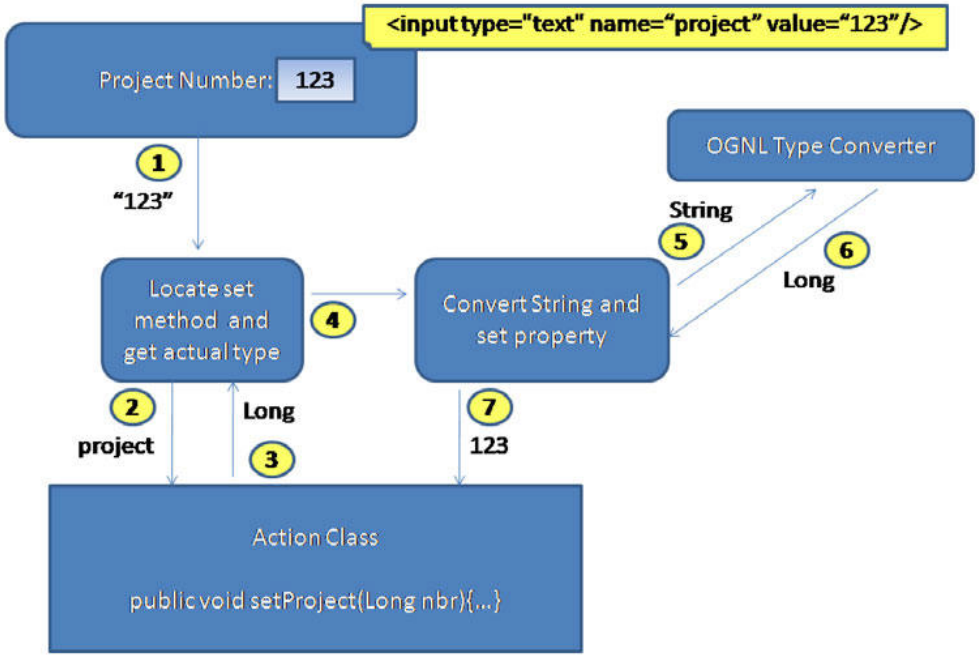
Struts 2 překládá a konvertuje všechny *String* parametry požadavku na komplexní datové typy nacházející se v konkrétní akci. Jakmile je napsán adaptér mapující web stránky na objekty modelu, je rázem ušetřeno mnoho hodin ručního psaní kódu. Struts 2 se také stará o překlad v opačném směru k uspokojení *String* požadavků HTML.

Překlad z uživatelských vstupů do business logiky ve Struts 2

Překlad mezi Stringy (vstupy uživatele) a modelem business logiky je uskutečňován podle OGNL pomocí *DefaultTypeConverter*. Ten je schopný pracovat s mnoha datovými typy, datumy, poli, mapami a kolekcemi.

Existuje samozřejmě možnost vytvořit si vlastní konvertor uživatelských datových typů. Děje se tak vytvořením konvertoru jako potomka třídy *StrutsTypeConverter* a registrací nového konvertoru buď jako samostatné třídy v *NazevTridy*-conversion.properties nebo globálně v xwork-conversion.properties.

Struts 2 plní objekty uživatelskými vstupy prostřednictvím **interceptoru**. Tento interceptor používá jméno vstupní kontroly webové stránky k vyhledání setter metody ve třídě akce. Jakmile metodu najde, rozhodne o druhu datového typu, na což setter metoda čeká a převede *String* na tento předem určený datový typ. Nakonec jsou konvertovaná data odevzdána setter metodě, která zajistí naplnění odpovídajícího objektu. Tento proces ukazuje následující obrázek 4.3.



Obr. 4.3: Překlad vstupů uživatele do business logiky ve Struts 2 (BROWN & DAVIS, 2008)

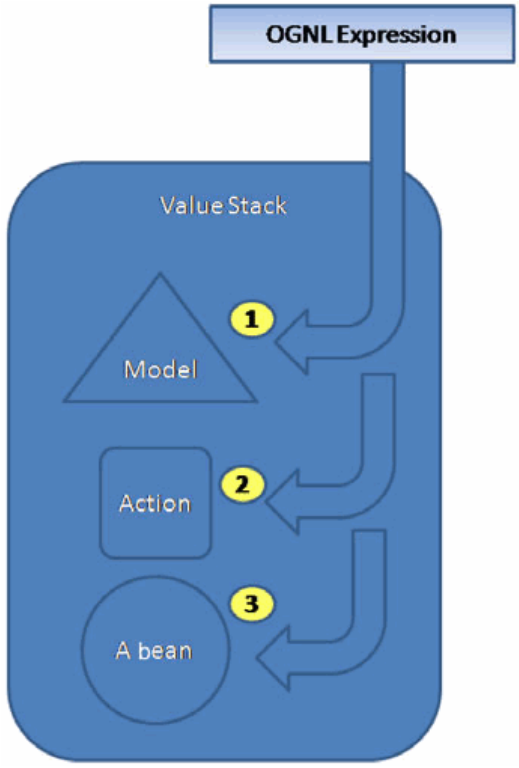
4.2.3 Knihovny tagů (Tag Libraries)

Knihovny tagů Strutsu 1 spolupracují s rámcem na řízení pohybu dat a pracovního postupu („workflow"). Struts 1 má několik knihoven tagů, jejichž činnosti se občas vzájemně překrývají. Struts 2 zefektivnil tento proces integrací všech knihoven do jedné. Mimojiné to velmi zjednodušuje rozhodování, jakou direktivu *taglib* je třeba deklarovat v záhlaví dané web stránky.

Největší funkční rozdíl mezi oběma rámci Struts je však v tom, že knihovny verze 1 závisí na zavrhnutém ActionFormu, kdežto verze 2 využívá OGNL *ValueStack*.

ValueStack ve Struts 2

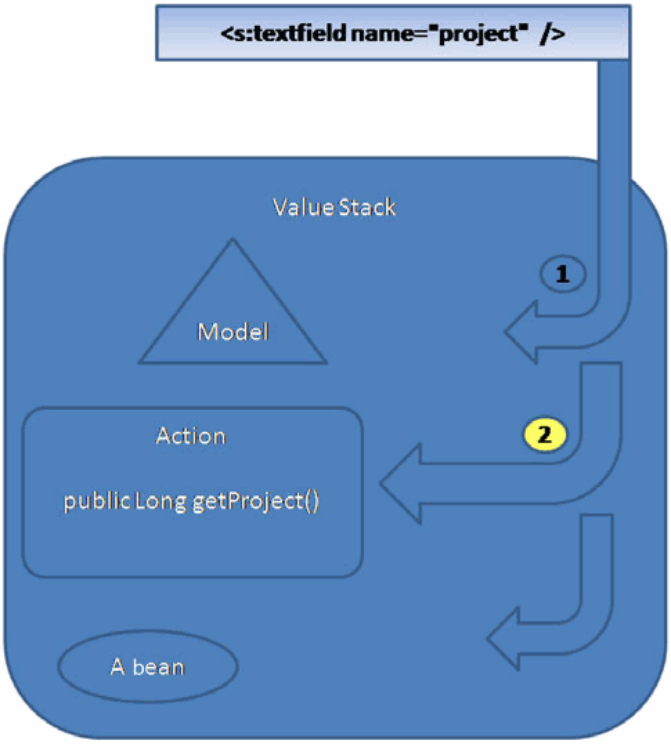
ValueStack je seřazený seznam real-time objektů. Funguje tak, že objekty jsou tlačeny dovnitř a vystrkovány ze zásobníku, tak jak framework vyřizuje požadavky. K tomuto pohybu jsou určeny speciální tagy jazyka OGNL. Jakmile je vyžadován konkrétní výraz (hodnota odpovídající původnímu uživatelskému vstupu), zásobník je prohledán odshora dolů. Tato flexibilnost umožňuje najít hledaný výraz, aniž by se nejprve muselo zjistit, který objekt danou vlastnost obsahuje. Viz obrázek 4.4.



Obr. 4.4: ValueStack - vyhledání hledaného výrazu (BROWN & DAVIS, 2008)

Jak lze vidět na tomto obrázku, OGNL hledá svůj výraz odshora zásobníku a pokračuje směrem dolů, dokud jej nenajde anebo nedosáhne konce zásobníku.

Následuje konkrétní příklad odpovídající předcházejícímu obecnému obrázku, který vysvětluje, jak je webová stránka v interakci s dynamickými objekty zásobníku. Další obrázek 4.5 ukazuje, jak ve Struts 2 uživatelská („custom") značka najde hodnotu *project*. V tomto případě byla nalezena v objektu Action. V porovnání se Struts 1 je tento mechanismus jazyka OGNL daleko efektivnější, neboť pro úspěšnou navigaci opravdu stačí najít správný řetězec u jakéhokoliv objektu v zásobníku, aniž by bylo třeba nejprve identifikovat objekt, který tuto hodnotu obsahuje.



Obr. 4.5: ValueStack - Konkrétní případ vyhledání výrazu (BROWN & DAVIS, 2008)

Jak již bylo výše uvedeno, přechod mezi oběma verzemi Struts vyžaduje nahrazení původních tag direktiv jedinou integrující: *<%@ taglib prefix="s" uri="/struts-tags"%>*.

Další novinkou je skutečnost, že tagy Struts 2 zahrnují automatickou internacionalizaci. To umožňuje odstranit mnoho HTML kódu, který musel být napsán pro zajištění internacionalizace pod Struts 1.

4.2.4 JSTL a Struts

Struts 1 disponoval několika knihovnami tagů, z nichž každá obsahovala související seskupení značek. Jakmile byla uvolněna technologie JSTL (JavaServer Pages Standard Tag Library), bylo doporučováno vybrat si JSTL verzi, která měla odpovídající značku v knihovně Struts 1. Bylo to z důvodu podpory JSTL pro nový Expression Language (umožňuje přístup JavaBeanům do JSP). JSTL Expression Language (EL) je sice bohatý a umožňuje proniknout hluboko do grafu objektů a má podporu pro práci s mapami a seznamy, ale Struts 2 má své vlastní knihovny tagů, které spolupracují s OGNL, jež má daleko větší možnosti použití než EL. Nicméně ve Struts 2 také existuje možnost využívat JSTL.

4.2.5 Internacionalizace

Java od svých prvních verzí vždy nabízela nativní podporu lokalizace. Struts 1, stejně jako Struts 2, se snaží převést nativní podporu z Javy přes třídy *java.util.PropertyResourceBundle* a *java.util.ListResourceBundle*. Rozlišnost obou verzí spočívá až ve větší flexibilitě při výběru lokalizovaného svazku.

Nyní je třeba ilustrovat, co výběr lokalizovaného svazku znamená na konkrétním příkladě: Existují určité webové stránky, které mají anglickou a francouzskou verzi. Musí tedy existovat soubory, které budou obsahovat příslušnou verzi. Budou to soubory: *Bundle_eng.properties* a *Bundle_fra.properties*. Výběr správného souboru záleží na jazyku požadavku uvedeného v HTTP hlavičce. Bude-li to francouzský požadavek, bude vybrán soubor *Bundle_fra.properties*. Další postup se již u obou verzí Struts liší.

Struts 1

Rámec očekává vše pro překlad stránek v souboru *Bundle_fra.properties*, což však není příliš šťastný přístup. Proto rámec umožňuje, aby byl všechen francouzský text stránek rozdělen ve více souborech. Tento postup ovšem není o moc efektivnější. Vyžaduje totiž uživatelskou Java třídu navíc a potřebu prefixovat všechny zprávy značkou souboru tak, aby tato třída starající se o řízení překladu stránek mohla určit, ze kterého souboru má příslušný text získat. To je velmi neobratný přístup náchylný k tvorbě chyb.

Struts 2

Struts 2 se liší v tom, že dává možnost variabilního rozsahu nastavení lokalizace počínaje určením souboru pro celou aplikaci (totožné se Struts 1) a konče na elementární úrovní jediné akce.

- **Na úrovni aplikace:**

Struts 2 funguje stejně jako Struts 1. Nabízí lokalizovaný svazek v *struts.properties* jako: *struts.custom.i18n.resources=resources.package.*

- **Na úrovni akce:**

Existují-li speciální lokalizační požadavky pro akci, např. *MemberAction*, je třeba vytvořit svazek (bundle) s názvem *MemberAction.properties* a umístit ho vedle třídy dané akce. Struts 2 posléze obdrží hodnoty právě z tohoto svazku.

Vyhledávací posloupnost při internacionalizaci aplikace ve Struts 2:

Tento postup definuje pořadí pro případy aplikace lokalizačních souborů pohybujících se mezi oběma krajními možnostmi (na úrovni aplikace a na úrovni akce).

1) *ResourceBundle* je určen stejným názvem i balíčkem jako třída objektu na zásobníku, v což jsou zahrnuty i veškerá implentovaná rozhraní ve třídě a rodičovská třída.

a) Hledat zprávu v *ResourceBundle* třídy.

b) Pokud nenalezeno, hledat v *ResourceBundle* v každém implementovaném rozhraní.

c) Pokud nenalezeno, skočit v hierarchii tříd na rodičovskou třídu a zopakovat krok a).

2) Pokud text zprávy nebyl nalezen ani po prohledání celé hierarchie tříd, ale zároveň objekt implementuje *ModelDriven*, je třeba volat metodu *getModel()* a prohledat hierarchii tříd objektu modelu.

3) Pokud zpráva stále není nalezena, je třeba hledat v hierarchii tříd defaultního balíčku.

a) Pro balíček původní třídy či objektu je hledán *ResourceBundle* pojmenovaný *package.properties* v balíčku.

b) Pro instanci, jestliže se třída jmenuje *com.strutsschool.enrollment.MemberAction*, je třeba v takových případech hledat *ResourceBundle* pojmenovaný jako *com.strutsschool.enrollment.package.properties*.

Dále se pokračuje stejně pro každou rodičovskou třídu v hierarchii.

4) Pokud Struts 2 dosud stále nic nenalezl, musí se zkontrolovat, zda klíč zprávy skutečně odkazuje na vlastnost objektu *ValueStacku*. Pokud je zjištěno, že *ValueStack* není prázdný a hledaný klíč je *member.course.description*, je třeba hledat řetězec *course.description* v celé hierarchii tříd stejně jako v předešlých krocích.

5) Posledním místem, které se prohledává, je defaultní *ResourceBundle*, který je registrován ve *struts.properties*.

Na první pohled vypadá vyhledávání potřebného řetězce takovým způsobem velmi komplikovaně, nicméně naučí-li se programátor pochopit tento postup a použít jej v kódu, může elegantně vyřešit i velmi komplikované požadavky na internacionalizaci webové aplikace. I když je nutno říci, že tato záležitost je jedna z těch, proč dost programátorů pokládá rámec Struts 2 za příliš komplikovaný. Zejména případný převod webové aplikace s rozsáhlou implementací internacionalizace a lokalizace ze Struts 1 na Struts 2 je v tomto ohledu velmi problematický.

4.3 Postup převodu aplikace ze Struts 1 do Struts 2

4.3.1 Zprovoznění rámce Struts 2 v aplikaci Struts 1

Při převodu aplikace je třeba nejprve zprovoznit Struts 2 v aplikaci následujícím postupem:

- Stáhnout ze stránek http://struts.apache.org/ aktuální verzi Struts 2 (v dubnu 2011 to byla verze 2.2.1.1).
- Zkopírovat ze Struts 2 obsah adresáře *struts-2.2.1.1\lib* do adresáře *WEB-INF/lib/* Struts 1 aplikace. Tímto způsobem přidáme do aplikace některé nezbytně nutné knihovny pro běh Struts 2.
- Zeditovat konfigurační soubor *web.xml* do podoby, jaká je na následujícím obrázku 4.6. Zde jde zejména o sladění URL mapování a směrování požadavků.
- Umístit soubory *struts.xml* (náhrada struts-config.xml) a *struts.properties* do adresáře classpath dané aplikace, což je typicky *WEB-INF/classes*. *Struts.properties* obsahuje množství parametrů, které mají zásadní vliv na běh frameworku.
  Oba soubory se nachází v instalačním balíku Struts 2 v adresáři
  struts-2.2.1.1\src\core\src\test\resources.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- pro přehlednost vypuštěna definice typu dokumentu -->
3  <web-app>
4  <!-- Struts 2 -->
5      <filter>
6          <filter-name>struts2</filter-name>
7          <filter-class>
8  org.apache.struts2.dispatcher.FilterDispatcher
9          </filter-class>
10     </filter>
11     <filter-mapping>
12         <filter-name>struts2</filter-name>
13         <url-pattern>/*</url-pattern>
14     </filter-mapping>
15
16 <!-- Struts 1 -->
17     <servlet>
18         <servlet-name>action</servlet-name>
19         <servlet-class>
20 org.apache.struts.action.ActionServlet
21         </servlet-class>
22         <init-param>
23          <param-name>config</param-name>
24          <param-value>/WEB-INF/classes/struts-config.xml</param-value>
25         </init-param>
26         <load-on-startup>2</load-on-startup>
27     </servlet>
28     <servlet-mapping>
29         <servlet-name>action</servlet-name>
30         <url-pattern>*.do</url-pattern>
31     </servlet-mapping>
32 </web-app>
```

Obr. 4.6: Soubor web.xml po začlenění filter sekce rámce Struts 2

Struts 2 je vložen do aplikace jako filtr, který umí zachytávat požadavky (řádky 6 až 11).

URL mapování je nastaveno na /*. URL končící **.action* bude spravovat Struts 2. Toto je předdefinováno v souboru *default.properties* u klíče *struts.action.extension*. Pro případnou změnu slouží totožný klíč v souboru *struts.properties* (řádky 12 až 15).

Struts 1 zůstává v aplikaci jako server (řádky 18 až 28).

URL mapování, o které se i nadále bude starat, je ve tvaru **.do* (řádky 29 až 32).

Od této chvíle je Struts 2 přidán do aplikace, takže je možno pod ním vytvářet nové funkcionality či převádět existující ze Struts 1.

Také je možné do web.xml zařadit další framework, a to přidáním kódu zobrazeného na následujícím obrázku 4.7.

```
1  <listener>
2      <listener-class>
3  org.apache.struts2.tiles.StrutsTilesListener
4      </listener-class>
5  </listener>
```

Obr. 4.7: Kód startující další rámec ve web.xml

Tento kód nastartuje jiný Framework, v tomto případě Tiles 2 (šablonovací rámec zjednodušující vývoj uživatelského rozhraní aplikace).

4.3.2 Mapování akcí

Mapování akcí se ve Struts 2 zásadně mění. Nejprve je užitečné rozebrat, jak mapování funguje ve Struts 1. Existuje v něm struts-config.xml, což je registr všech symbolických adres webové aplikace a jejich propojení se svými protějšky. Následující obrázek 4.8 ukazuje typický struts-config.xml (pro přehlednost je vypuštěna hlavička souboru).

```
1  <struts-config>
2      <form-beans>
3          <form-bean name="reportForm" type="ReportForm"/>
4      </form-beans>
5      <action-mappings>
6          <action
7  path="/reportDateSelection"
8  input="/reportDateSelection.page"
9  name="reportForm"
10 type="ReportDateSelection"
11 scope="request">
12             <forward name="success" path="reportByDate.page"/>
13             <forward name="largeDateRange" path="reportWarning.page"/>
14         </action>
15     </action-mappings>
16 </struts-config>
```

Obr. 4.8: Struts-config.xml rámce Struts 1

Nejdříve se musí vytvořit form bean, který je oddělen od samotné akce. Tomuto beanu je přiřazeno jméno a může být asociován s mnoha odlišnými akcemi (řádek 2, 3, 4).

ReportDateSelection je třída akce asociovaná s tímto definovaným mapováním (řádek 10).

Do atributu *path* je uložena symbolická adresa dané webové funkcionality (řádek 7).

Do atributu *input* je uložena cesta ke stránce, jež se má zobrazit v případě neúspěšné validace (řádek 10).

Atribut *name* je propojení na výše definovaný form bean (řádek 9).

Atribut *scope* určuje místo, kde má být form bean uložen (řádek 11).

Oba tagy *forward* určují cíle, které přicházejí v úvahu pro danou akci (řádek 12, 13).

Web požadavek *someContex/reportDateSection.do* vyvolá mapování akcí ve Struts 1. Struts 2 zjednodušuje mapování využitím ukládání uživatelských vstupů v akci samotné. Následující obrázek 4.9 ukazuje soubor struts.xml (náhrada struts-config.xml), ve kterém je řešeno mapování akcí podle Struts 2 (opět je pro přehlednost vypuštěna hlavička souboru). Struts.xml se umísťuje do adresáře *WEB-INF/classes* aplikace.

```
1  <struts>
2    <package name="invoicing" namespace="/invoicing">
3       <action name="reportDateSelection" class="ReportDateSelection">
4           <result name="input">reportDateSelection.page</result>
5           <result name="largeDateRange">reportWarning.page</result>
6           <result>reportByDate.page</result>
7       </action>
8    </package>
9  </struts>
```

Obr. 4.9: Struts.xml rámce Struts 2

V mapování se objevuje tag *package*. Pracuje podobně jako balíček v Javě, kdy podobné akce jsou seskupeny dohromady podle běžného jmenného prostoru. Web požadavek *someContext/invoicing/reportDateSelection.action* vyvolá tuto Struts 2 akci. Tento princip umožňuje použít stejné jméno akce v jakémkoliv balíčku jmenného prostoru (řádek 2).

Samotné mapování akce bylo oproti Struts 1 velmi zjednodušeno. Byl zrušen form bean, Struts 2 zapouzdřuje vstupy uživatele dovnitř samotné akce (řádek 3).

A konečně všechny cíle, které přicházejí pro danou akci v úvahu, jsou ponechány uvnitř akce ve značkách result. Poslední tag *result* nemá specifikováno jméno, protože akce *success* je výchozí (řádek 4, 5, 6).

Ve Struts 1 vstupy uživatele byly uloženy ve form beanu, které rámec předal do třídy akce. Struts 2 zanořuje tento form do akce samotné, takže samostatný form bean nevytváří. Takže prvním krokem převodu Struts 1 na Struts 2 musí být přizpůsobení Struts 1 akcí při umísťování uživatelových vstupů. Stačí tedy odstranit kód z form beanu a vložit jej jako POJO do třídy akce. Tento přesun ukazují následující obrázky 4.10 a 4.11.

```
1   public class ReportForm extends ActionForm {
2
3       private boolean average;
4       private boolean totals;
5       private String fromDate;
6       private String toDate;
7
8       /** kód dále pokračuje implementací metod get/set/is */
```

Obr. 4.10: Struts 1 form bean před převodem

Tento Struts 1 form bean je potomkem třídy ActionForm (řádek 1) a je složen z primitivních datových typů (řádek 3 až 6).

```
1   import java.util.Date;
2   public class Report {
3
4       private boolean average;
5       private boolean totals;
6       private Date fromDate;
7       private Date toDate;
8
9       /** kód dále pokračuje implementací metod get/set/is */
```

Obr. 4.11: Struts 2 bean po převodu

Tento obrázek ukazuje Struts 2 verzi původního Struts 1 form beanu po převodu. Tato nová verze beanu je už POJO, který je naplněn užitečnými datovými typy obsahujícími vstupy uživatele. V případě vstupu uživatele typu Datum již nejsou zaváděny Stringy jako u Struts 1, ale už přímo datový typ Date.

Následující obrázek 4.12 zobrazuje třídu akce Struts 2. Je zde vidět, jak jsou zpřístupněna uživatelská data.

```
1   public class ReportDateSelection extends ActionSupport {
2
3       private Report model;
4       private String SUCCESS;
5
6       public String execute() {
7   // zde bude business logika ...
8           return SUCCESS;
9       }
10      public Report getModel() {
11          return model;
12      }
13      public void setModel(Report model) {
14          this.model = model;
15      }
16  }
```

Obr. 4.12: Struts 2 akce s modelem bean

Třída Struts 2 *ReportDateSelection* je potomkem třídy Struts 2 *ActionSupport* (řádek 1).

Protože třídy akce již nejsou jedináčky jako v případě Struts 1, můžeme zavést proměnnou instance, jež přestavuje POJO model a poskytuje vstupy uživatele. Webová data jsou tedy zapouzdřena do model beanu. (řádek 2).

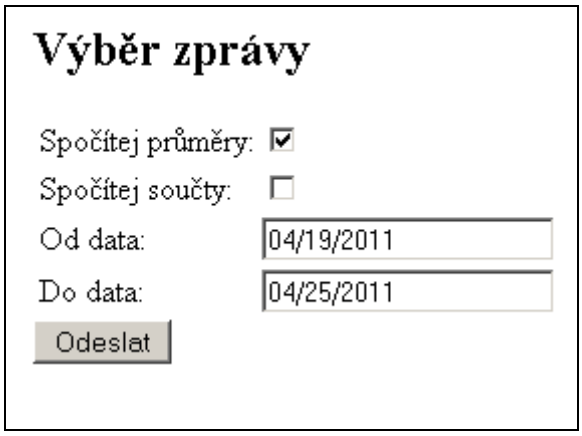
Metody *get/setModel()* využívá rámec pro udržení svého ValueStacku synchronizovaného se vstupy uživatele.

4.3.3 Převod webových stránek

Tato část převodu Struts 1 na Struts 2 zřejmě bude zpravidla zabírat nejvíce času. Tagy Struts 1 jsou v podstatě schopné pouze přepínat obsah webových stránek. Tagy Struts 2 jsou lepší v tom, že jsou schopny pomáhat i při vlastní prezentaci stránek.

Uživatelské tagy Struts 1 pomáhají oddělit kód Javy od kódu prezentace webových stránek. Uživatelské tagy Struts 2 však dokáží nejen totéž, ale umí silně zestručnit HTML kód, který by vývojář Struts 1 měl v úmyslu napsat.

Následující obrázek, Obr. 4.13, ukazuje webovou stránku, ke které pro názornost úspory kódu bude uveden jednak zdrojový kód napsaný pod Struts 1, tak i zdrojový kód napsaný pod Struts 2.



Obr. 4.13: Web stránka „Výběr zprávy“

Dalším obrázkem bude výpis zdrojového kódu webové stránky Výběr zprávy ve Struts 2. Viz obrázek 4.14.

```
1  <%@ taglib prefix="s" uri="/struts-tags"%>
2  <H3><s:label key="promptTitle" /></H3>
3  <s:form>
4      <s:checkbox key="average" labelposition="left"/>
5      <s:checkbox key="totals" labelposition="left"/>
6      <s:textfield key="fromDate" />
7      <s:textfield key="toDate"/>
8      <s:submit key="ok" action="reportParameters" />
9  </s:form>
```

Obr. 4.14: Web stránka „Výběr zprávy“ – kód Struts 2

Následuje obrázek, Obr. 4.15, ukazující tutéž stránku zapsanou ve Struts 1

```
<%@taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<H3> <bean:message key="promptTitle" /> </H3>
<BR>
<html:errors />
<html:form action="reportParameters">
    <TABLE border="0" cellpadding="0">
        <TBODY>
            <tr>
                <TD align="right">
                    <b><bean:message key="average" /></b>
                </TD>
                <TD>
        <html:checkbox property="average"></html:checkbox>
        </TD>
        </tr>
        <!-- zde je pro zestručnění vynecháno 20 řádků kódu-->
        <tr>
            <TD align="right">
                <b><bean:message key="toDate" /></b>
            </TD>
            <TD>
        <html:text property="toDate"></html:text>
        </TD>
        <TD>
        <html:errors property="toDate" />
        </TD>
        </tr>
        </TBODY>
    </TABLE>
    <html:submit property="submit">
        <bean:message key="odeslat" />
    </html:submit>
</html:form>
```

Obr. 4.15: Web stránka „Výběr zprávy“ – kód Struts 1

.

Jak je patrné z výše uvedených obrázků, Struts 2 stačí oproti Struts 1 v tomto případě šestkrát méně řádků kódu k zobrazení totožné stránky. Je to způsobeno zejména tím, že tagy ve Struts 2 samy generují některé standardní značky HTML. K tomuto postupu využívá Struts 2 FreeMarker (Java nástroj ke generování textového výstupu v HTML apod.). Pro detailní výčet všech tagů Struts 2 je uveden tento odkaz: http://struts.apache.org/2.0.14/docs/tag-reference.html, jelikož popis všech tagů by přesahoval rozsah této práce.

4.3.4 Internacionalizace

Struts 1 i Struts 2 podporují stejné Java internacionalizační funkcionality. V případě Struts 1 jsou specifikovány lokalizační soubory ve *struts-config.xml* tímto tagem:

<message-resources parameter="applicationResources" />

Tento existující *ResourceBundle* může být použit ve Struts 2 umístěním následujícího řádku do konfiguračního souboru struts.properties:

struts.custom.i18n.resources= applicationResources

Tento řádek ukazuje, že existuje *ResourceBundle* s root jménem *application-Resources* a alternativní soubor pojmenovaný *applicationResources_es.properties*, kde *es* znamená standardní kód pro španělštinu. *ResourceBundle* i soubor *struts.properties* by měl být umístěn do adresáře tříd webové aplikace, takže by měl být dostupný přes cestu třídy. Pokud bylo zapotřebí využít všechny možnosti internacionalizace pod Struts 2, lze odkázat na kapitolu 4.2.5 Internacionalizace této práce. Následující obrázek 4.16, ukazuje lokalizované soubory stránky, jež byla popisována v minulé podkapitole týkající se převodu stránek. Jedná se tedy o stránku Výběr zprávy (viz obrázek 4.13).

```
### applicationResources_cz.properties
promptTitle= Výběr zprávy
average=Spočítej průměry
totals=Spočítej součty
type=Typ zprávy
fromDate=Od data
toDate=Do Data
ok=Odeslat

### applicationResources_es.properties
promptTitle= Informe de selección
average= Promedios del cálculo
totals= Totales del cálculo
type= Tipo de informe
fromDate= A partir de fecha
toDate= Hasta la fecha
ok=Enviar
```

Obr. 4.16: Lokalizované soubory – čeština a španělština

Struts 1 třída akce obdrží tuto lokalizační zprávu:

getResources(request).getMessage("promptTitle")

Struts 2 je stručnější:

getText("promptTitle")

Toto je další kontrast mezi přístupem Struts 1 ke správě objektů a Struts 2 technologií Dependency Injection (metoda popsaná v kapitole 2.8.2.1 Inversion of Control). Metoda *getText()* Struts 2 se totiž může tázat požadavku jako svého vlastního. Dalším rozdílem je skutečnost, že když je ve Struts 1 dotazována zpráva, jejíž klíč nemůže být nalezen ve zdrojovém souboru, je v odpovědi pouze *null*. Struts 2 v tomto případě vrátí klíč indikující chybějící zprávu.

4.3.5 Validace

Jak již bylo řečeno dříve v kapitole 2.8.1 Apache Struts, ve Struts 1 se o kontrolu vstupních dat stará Commons Validator, který se opírá o třídy *ValidatorForm* a *ValidatorActionForm*, jež jsou odvozeny ze třídy ActionForm. Je dosti komplikované vůbec naučit spolupracovat CommnonValidator se Struts 1. Další problém většinou nastává ve chvíli, kdy jedno z API použitých při tvorbě aplikace prošlo nějakým významnějším vývojem. Potom většinou nezbývá, než celou validaci přepsat od začátku.

Validační funkcionalita ve Struts 2 je přímo komponenta jádra rámce, jež byla vyvinuta z OpenSymphony XWork (jiný framework pro webové aplikace, nabízející komplexní služby, nejen validaci). Tato komponenta nabízí všechny funkcionality Commnons Validator Struts 1, ale mnohem jednodušší formou.

Rozdíl ve validaci obou verzí bude demonstrován opět na webové stránce Výběr zprávy (obrázek 4.13). Předpokladem je, že Common Validator již byl úspěšné nakonfigurován a připojen do **Struts 1**. Následující obrázek 4.17 ukazuje implementaci validačních pravidel v konfiguračním souboru validations.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <!DOCTYPE form-validation PUBLIC
3  "-//Apache Software Foundation//DTD Commons Validator Rules
4  Configuration 1.3.0//EN"
5  "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd ">
6  <form-validation>
7      <formset>
8          <form name="S1ReportForm">
9              <field property="fromDate" depends="required,date">
10                 <arg0 key="fromDate"/>
11             </field>
12             <field property="toDate" depends="required,date">
13                 <arg0 key="toDate"/>
14             </field>
15         </form>
16     </formset>
17 </form-validation>
```

Obr. 4.17: Soubor validations.xml rámce Struts 1

Tag <*formset*> definuje skupinu formulářů (řádek 7).

Na dalším řádku název formuláře odpovídá Struts 1 bean jménu (řádek 8).

Každý tag <*form*> obsahuje validační pravidla každého konkrétního volání (řádek 9, 12), tento mechanismus je typický pro Struts 1. U složitějších aplikací bývá výsledkem obrovský soubor validations.xml, který je složité administrovat.

Jestliže validační pravidlo obsažené uvnitř tagu <form> neschválí vstup uživatele jako korektní, je odeslána chybová hláška webovému prohlížeči obsahující hodnotu parametru key (řádek 10 a 13).

Struts 2 nabízí jednodušší řešení validace. Struts 2 řídí totiž sofistikovaněji validační pravidla na úrovni objektu, místo aby shromažďovala validace celé webové aplikace v jednom souboru jako Struts 1. Rozdíl mezi oběma rámci je jako obvykle v tom, že Struts 1 problém koncentruje do jednoho místa a snaží se jej řídit jako celek. Struts 2 problém rozparceluje na elementární části, se kterými se mu pak daleko jednodušeji pracuje. Existuje mnoho pravidel, jak dosáhnout potřebné úrovně „zrnitosti“ („granularity"), běžným přístupem je jmenná konvence ve formátu *{Action}*-validation.xml.

Struts 2 podporuje ve validaci na straně uživatele kontrolu přes technologie JavaScript či AJAX.

Na následujícím obrázku 4.18, je zobrazena validace akce Report zobrazenou na předcházejícím obrázku 4.12 (Struts 2 akce s modelem bean).

```
1   <!DOCTYPE validators PUBLIC
2   "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
3   "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
4   <validators>
5       <field name="model.fromDate">
6           <field-validator type="required">
7               <message key="required" />
8           </field-validator>
9           <field-validator type="date">
10              <message key="date" />
11          </field-validator>
12      </field>
13  </validators>
```

Obr. 4.18: ReportDateSelection-validation.xml rámce Struts 2

Tag <*field*> definuje skupinu <*field-validator*> tagů (řádek 5).

Každý z těchto <*field-validator*> (řádek 6, 9) kontroluje dle vlastních pravidel.

Jestliže konkrétní kontrola selže, hodnota atributu key tagu <*message*> (řádek 7) je zobrazena ve webovém prohlížeči jako chybová hláška.

Zprávy (nejen ty chybové) ve Struts 2 mohou být konstruovány jako proměnlivá data vytažená přímo ze zásobníku *ValueStack* ve vztahu ke konkrétnímu požadavku. To umožňuje vytvářet pro uživatele přesně cílené zprávy rychle a inteligentně, bez potřeby složitých dynamických konstrukcí uvnitř zprávy.

4.3.6 Zásuvný modul Struts 1

Struts 2 nabízí zásuvný modul („plugin"). Jeho smyslem je použít třídy akcí napsané ve Struts 1 tak, aby se dané akce daly použít i pod Struts 2. Jelikož plugin umožňuje Struts 1 akcím používat i své ActionFormy, je přístupná i standardní Struts 1 validace těchto akcí pomocí Commons Validatoru. Plugin se jmenuje *struts2-struts1-plugin* a je třeba jej umístit do adresáře WEB-INF/lib webové aplikace. Plugin umožňuje Struts 2 interceptorům přijmout životní cyklus požadavku Struts 1 akce.

Ve Struts 1 je každá třída akce jedináčkem a je v interakci s form bean v rozsahu, jaký programátor specifikuje. Struts 2 nezná koncept form beanu. Takže úkolem pluginu je, aby předstíral, že i přejaté akce Struts 1 jsou akcemi Struts 2.

Plugin vytváří nový Struts 2 balíček se jménem *struts1-default*, který rozšiřuje *struts-default*. Nový *struts1-default* balík obsahuje nové interceptory, které jsou nabaleny na defaultní interceptor. Mají za úkol předstírat životní cyklus Struts 1 požadavku. Následující obrázek 4.19, ukazuje, jakým způsobem se konfiguruje Struts 1 akce, aby byla provozuschopná i ve Struts 2 aplikaci (BROWN & DAVIS, 2008).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="hybridActions"
namespace="/old2new"
extends="struts1-default">
   <action name="enroll" class="org.apache.struts2.s1.Struts1Action" >
       <param name="className">
           com.strutsschool.s1.actions.EnrollAction
       </param>
       <interceptor-ref name="scopedModelDriven" >
           <param name="className">s1.webapp.forms.EnrollForm</param>
           <param name="name">enrollForm</param>
           <param name="scope">request</param>
       </interceptor-ref>
       <interceptor-ref name="struts1Stack"></interceptor-ref>
       <result name="success">enrollPage</result>
   </action>
    </package>
</struts>
```

Obr. 4.19: Převod akcí Struts 1, aby běžely ve Struts 2

Tag <*package*> je typicky nakonfigurován se jmenným prostorem a rodičovským balíčkem (řádek 6, 7, 8).

Struts 1 třída zásuvného modulu (řádek 9) očekává parametr (řádek 10, 11, 12), který popisuje jméno existující Struts 1 třídy akce.

Následující kód konfiguruje *scopedModelDriven* interceptor. Tento interceptor je nedílnou součástí Struts 2 a má společné rozhraní stejného jména (řádek 13).

Struts 1 třída implementuje toto rozhraní, takže parametry interceptoru *className* (řádek 14), *name* (řádek 15) a *scope* (řádek 16) vlastně specifikují Struts 1 action form.

Poslední interceptor definuje zásobník interceptorů, které vykonávají tento celý postup (řádek 18).

5 Závěr

Práce poskytla přehled nejoblíbenějších a nejvyužívanějších MVC rámců prostředí Java EE. Vysvětlila principy fungování a postupy tvorby jak požadavkem řízených, tak modernějších, komponentně orientovaných rámců. Po přečtení této práce by měl pokročilejší vývojář platformy Java EE pochopit, pro které oblasti je příslušný rámec vhodný. Na základě praktických kritérií se pokusila naznačit, ve které oblasti bude každý rámec jistější, jakým způsobem přistupuje k testování, validování, konverzi, internacionalizaci a další funkcionalitám, které jsou dnes od MVC rámce očekávány. Rozsah této práce bohužel neumožňuje hlubší pochopení daného rámce, na to je potřeba přece jen více místa. Nicméně jako základní vodítko by měla posloužit.

Hlavním přínosem práce jsou kapitoly 3 a 4 poskytující podporu rozhodování, zda vůbec a jakým způsobem převést webovou aplikaci napsanou ve Struts 1 na Struts 2. Pro aplikace, u nichž se již nepředpokládá rozšiřování funkcionalit, jednoznačně přechod na modernější rámec nemá smysl. V případě firmy CompuGroup Medical, zabývající se dlouhodobě nemocničními informačními systémy, je předpoklad dlouhé existence a možné úpravy funkcionalit webových aplikací založených na Struts 1, které již byly implementovány a budou udržovány, i v budoucnu. Tudíž klady převodu aplikace (větší efektivnost tvorby s modernějším rámcem a možnost efektivnějšího testování, hrozba ukončení podpory Struts 1) jednoznačně převažují nad riziky (podcenění příprav přechodu - zejména jeho podfinancování).

V neposlední řadě by mohla být užitečná poslední kapitola práce, která je určena pro vývojáře (nejen firmy CompuGroup Medical), kteří mají zvládnutý Struts 1, ví o jeho slabinách a chtěli by přejít na modernější a efektivnější rámec. Takovým je právě Struts 2, v jeho případě je přínosem velká příbuznost se svým předchůdcem. Jsou zde nastíněna hlavní úskalí, která na programátory čekají. Doba jednoznačně nazrála k modernizaci rámce. Struts 1 pomalu ztrácí své dominantní postavení, i když se zdá, že je stále jedničkou v rozšířenosti mezi MVC rámci.

Seznam použité literatury

Apache Software Foundation, The. *Struts* [online]. 2011. Dostupný z WWW: <http://struts.apache.org/>.

Apache Software Foundation, The. *Tapestry* [online]. 2011. Dostupný z WWW: <http://tapestry.apache.org/>.

BROWN, D.; DAVIS, C. M.; STANLICK, S. *Struts 2 in Action*. 1. vyd. Greenwich: Manning, 2008. 432 s. ISBN 1-933988-07-X.

CAVANES, C. *Programming Jakarta Struts*. 2. vyd. Sebastopol: O'Reilly Media, 2004. 550 s. ISBN 0-596006-51-9.

Dice Holdings, Inc. (firma). *Dice : The career hub for tech* [online]. c2011 [cit. 2011-04-25]. Dostupný z WWW: <http://www.dice.com/>.

FALKNER, J.; JONES K. *Servlet and JavaServer Pages: The J2EE Technology Web Tier.* 1. vyd. Boston: Addison-Wesley, 2003. 758 s. ISBN 0-321-13649-7.

GEARY D.; HORSTMANN, C. S. *Core JavaServer Faces*. 2.vyd. Boston: Prentice Hall, 2007. 727 s. ISBN 0-13-173866-7.

MANN, K. D. *JavaServer Faces in Action.* 1. vyd.: Greenwich Manning, 2004. 744 s. ISBN 1932394125.

MoroSystems, s.r.o. (firma). *VsadNaJavu.cz : je odborný weblog firmy* [online]. c2011 [cit. 2011-04-20]. Tapestry 5 – úvod. Dostupný z WWW: <http://vsadnajavu.cz/2009-05/java-j2ee/tapestry-5-uvod/>.

MAHDÁL, J. *Moderní rámce pro tvorbu webových aplikací*. Brno, 2006. 63 s. Diplomová práce (Ing.). Masarykova univerzita, Fakulta informatiky. Dostupný také z WWW: < http://is.muni.cz/th/3587/fi_m/diplomka.pdf?lang=en>.

NOVÁKOVÁ, P.. *Komponentně orientované webové rámce*. Brno, 2008. 64 s. Diplomová práce (Ing.). Masarykova univerzita, Fakulta informatiky. Dostupný také z WWW: <http://is.muni.cz/th/42408/fi_m_a2/diplomova_prace.pdf>.

RAIBLE,M. *Raible Designs* [online]. c2011 [cit. 2011-04-22]. Comparing JVM Web Frameworks. Dostupný z WWW: <http://raibledesigns.com/rd/entry/my_comparing_jvm_web_frameworks>.

Rose India Technologies Pvt. Ltd. (firma). *www.roseindia.net : JSF Life Cycle* [online]. c2011 [cit. 2011-04-22]. JSF Life Cycle. Dostupný z WWW: <http://www.roseindia.net/jsf/jsflifecycle.shtml>.

SEVRJUKOV, A. *PVC : PHP Model View Controller framework* [online]. c2007 [cit. 2011-04-22]. Co je Model View Controller? Dostupný z WWW: <http://pvc.boolean.cz/?section=basic&page=mvc>.

SHERWOOD. NIS CGM G3 Systema *CGM - CompuGroup Medical* [on-line]. 2011 [cit. 2011- 04-23]. Dostupné z: <http://www.compugroup.cz/nemocnice/nemocnicni-informacni-systemy-nis/cgm_g3_systema/>.

SpringSource, a division of VMware. *SpringSource* [online]. 2011. Dostupný z WWW: <http://www.springsource.com/>.

TRONÍČEK, Z. JavaServer Faces: vytváříme komponentu *java.cz* [on-line]. 11. 2. 2009 [cit. 2009- 04-21]. Dostupné z: <http://www.java.cz/article/18718/>.

VONDROUŠ, J. *Renovace MVC frameworků*. Brno, 2008. 56 s. Diplomová práce (Ing.). Masarykova univerzita, Fakulta informatiky. Dostupný také z WWW: <http://is.muni.cz/th/99193/fi_m/dp.pdf?lang=cs>.

WALLS, C.; BREIDENBACH, R. *Spring in Action*. 1. vyd. Greenwich: Manning, 2005. 472 s. ISBN 1-932394-35-4.

Seznam zkratek

AJAX	Asynchronous JavaScript and XML: Technologie umožňující měnit obsah svých stránek bez znovunačítání.
AOP	Aspect-oriented programming. Podpora aspektově orientovaného programování.
API	Application Programming Interface. Rozhraní pro programování aplikací.
CGM D3 Systema	Nemocniční IS firmy CGM, 3. generace (tenký klient).
CSS	Cascading Style Sheets. Jazyk pro definici pravidel formátování internetových stránek.
DAO (modul)	Data Access Object. Modul umožňující přistupovat k databázím různých výrobců.
EJB	Enterprise JavaBeans. Jsou řízené, serverové komponenty umožňující modulární tvorbu podnikových aplikací (součást Java EE).
EL	Expression Language. Skriptovací jazyk umožňující přistupovat k Java objektům přes JSP stránky
G2 Java	Aplikace napsaná v Javě, 2. generace, tlustý klient.
G3 Adobe Flash	Aplikace napsaná v Adobe Flash, 2. generace, tenký klient.
GIF	Graphics Interchange Format. Grafický formát určený pro rastrovou grafiku, zejména pro zobrazení přes WWW.
GUI	Graphical User Interface. Uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků.
HL7	Health Level Seven International. Mezinárodní standard pro součinnost ve zdravotnických IT.
HTML	HyperText Markup Language. Jazyk pro vytváření stránek v systému World Wide Web, který umožňuje publikaci dokumentů na Internetu.
HTTP	Hypertext Transfer Protocol. Internetový protokol určený pro výměnu hypertextových dokumentů ve formátu HTML.
i18n	Internacionalization. Zkratka pro slovo internacionalization, které mezi prvním ("i") a posledním písmenem ("n") obsahuje 18 znaků.
ID	Identification. Identifikace ve výpočetní technice.
IDE	Integrated Development Environment. Software usnadňující práci programátorů, většinou zaměřený na jeden programovací jazyk.
IHE	Integrating the Healthcare Enterprise. Mezinárodní standard pro sdílení informací ve zdravotnictví.
IoC	Inversion of Control. Princip většiny moderních komponentně orientovaných rámců.
JAR	**J**ava **AR**chive. Jedná se o archivní soubor pro programovací jazyk Java založen na zip kompresi.
Java EE	Java Platform, Enterprise Edition. Součást platformy Java určená pro vývoj a provoz podnikových aplikací a informačních systémů.
JDBC	Java Database Connectivity. API pro Java programátory, které definuje jednotné rozhraní pro přístup k relačním databázím.
JDK	Java Development Kit. Soubor základních nástrojů pro vývoj aplikací pro platformu Java.
JDO	Java Data Objects. Specifikace Javy pro trvalost objektů.
JNDI	Java Naming and Directory Interface. Standardní rozšíření Java platformy, rozhraní pro práci s různými jmennými a adresářovými službami.

JSF	Java Server Faces. Komponentně orientovaný MVC rámec.
JSP	Java Server Pages. Java technologie pro tvorbu dynamických web stránek založených na HTML, XML a jiných dokumentech.
JSTL	JavaServer Pages Standard Tag Library. Součást Java EE. Rozšiřuje JSP o knihovnu značek pro běžné účely (např. internacionalizace).
l10n	Localization. Zkratka pro slovo localization, které mezi prvním ("l") a posledním písmenem ("n") obsahuje 10 znaků.
MVC	Model-View-Controller. Druh web architektury založené na jednoznačném oddělení prezentační a logické složky.
NIS	Nemocniční informační systém. IS integrující veškerý nemocniční software do jedné platformy.
OGNL	Object Graph Notation Language. Jazyk pro definování vazeb mezi atributy tříd modelu a komponentami na straně uživatele.
ORM	Object-relational mapping. Programovací technika pro konverzi mezi nekompatibilními datovými typy v objektově orientovaném jazyku.
PDF	Portable Document Format. Souborový formát vyvinutý firmou Adobe pro ukládání dokumentů nezávisle na softwaru i hardwaru, na kterém byly pořízeny.
POJO	Plain Old Java Object. Obyčejný Java object, není žádným speciálním objektem, zejména ne Enterprise JavaBeanem.
PSČ	Poštovní směrovací číslo.
SQL	Structured Query Language (strukturovaný dotazovací jazyk). Standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích.
TLD	Tag Library Descriptor. Soubor popisující vlastnosti a umístění Tag class.
URL	Uniform Resource Locator (jednotný lokátor zdrojů). Řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací.
ÚVN	Ústřední vojenská nemocnice.
WAR	**W**eb application **AR**chive. JAR archiv určený k distribuci web aplikace (JSP + servlety + Java třídy + XML dokumenty, ...).
WML	Wireless Markup Language. Značkovací jazyk založený na jazyce XML umožňující tvorbu online dokumentů pro mobilní zařízení.
XML	Extensible Markup Language. Umožňuje snadné vytváření konkrétních značkovacích jazyků (tzv. aplikací) pro různé účely a různé typy dat.
XSLT	e**X**tensible **S**tylesheet **L**anguage **T**ransformations. Transformace sloužící k převodům zdrojových XML dat do jiného formátu, např. HTML.

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst.3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB- TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 29. dubna 2011

Vít Strakoš
jméno a příjmení studenta

Adresa trvalého pobytu studenta:
Bystřička 117, 756 24 Bystřička

Seznam příloh

obsah přiloženého CD:

/text – tato práce ve formátech docx a pdf

/install – zip archiv „struts-2.2.1.1-all.zip“ obsahující vše potřebné k instalaci Struts 2